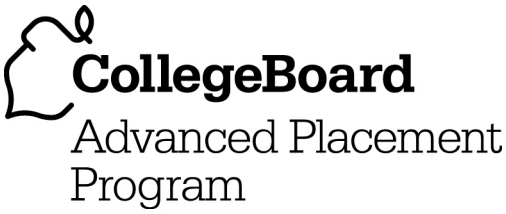


# AP® Computer Science AB
# 2004 Scoring Guidelines

# AP® Computer Science AB
# 2004 SCORING GUIDELINES

## Question 1

**Part A:** `LibraryItem` **2 pts**

**+1/2** `public interface LibraryItem` (missing public ok but private loses the point)
**+1/2** `String id()`
**+1/2** `String holder()`
**+1/2** `void setHolder(String holder)`

Note: No penalty for extra methods, except set id loses ½ usage
Note: Method identifiers must be appropriate or lose the ½ point.
Single character identifier ok for parameter only.
Note: ½ point usage each, for including data fields or constructors or implementing methods
(taken only once for each of these three, max 1 ½)

**Part B:** `LibraryBook` **4 pts**

**+1** `public class LibraryBook extends Book implements LibraryItem`
- **+1/2** attempt (must extend Book or implement LibraryItem)
- **+1/2** correct

**+1/2** `private String` instance variables with appropriate identifiers for id and holder

**+1** Constructor
- **+1/2** attempt (must use `super` or correctly set ID )
- **+1/2** correct

**+1 1/2** methods, identifiers must match identifiers in part a, bodies must be correct
- **+1/2** providing functionality for `id`
- **+1/2** providing functionality for `holder`
- **+1/2** providing functionality for `setHolder`

Note: ½ point usage if full interface from part A is not implemented.

**Part C:** data structure **3 pts**

**+1 1/2** Specify data structure
- **+1** choose `HashMap` or `TreeMap`
- **+1/2** key is ID (must be `Map`)

**+1 1/2** complexity (eligible for these points for any data structure choice, as in table below)
- **+1/2** `add`
- **+1/2** `checkOut`
- **+1/2** `getHolder`

| | unsorted AL/ LL | sorted AL | sorted LL | HashSet | TreeSet | HashMap | TreeMap |
|---|---|---|---|---|---|---|---|
| `add` | O(1) | O(N) | O(N) | O(1) | O(logN) | O(1) | O(logN) |
| `checkOut` | O(N) | O(logN) | O(N) | O(N) | O(N) | O(1) | O(logN) |
| `getHolder` | O(N) | O(logN) | O(N) | O(N) | O(N) | O(1) | O(logN) |

# AP® Computer Science AB
# 2004 SCORING GUIDELINES

## Question 2

**Part A:** `VoterBallots constructor` **4 pts**

**+1/2** create `Map`

**+1** loop through `ballotList`
- **+1/2** attempt
- **+1/2** correct

**+1** iterate through each ballot Set (in context of `ballotList` loop)
- **+1/2** attempt (must have 2 of decl/retrieval, hasNext, next)
- **+1/2** correct

**+1 1/2** create or update entry (eligible for these points even if item from `ballotList` is used for candidate)
- **+1/2** check if candidate is in `Map` as a key (lose this if options are reversed)
- **+1/2** conditionally, increment count
- **+1/2** conditionally, create entry with count of one

**Part B:** `candidatesWithMost` **4 pts**

**+1/2** create and return result `Set`

**+1/2** call `maxVotes` (must store or use returned value)

**+1** Iterate through `Map` key set
- **+1/2** attempt
- **+1/2** correct

**+1** check if count from `Map` equals max
- **+1/2** attempt (must test equality of something from `Map` to some max)
- **+1/2** correct (must be in context of loop)

**+1** Adjust result set appropriately

**Part C:** complexity **1 pt**

**+1** correct complexity

for HashMap and HashSet, O(C)
for HashMap and TreeSet, O(C log C)
for TreeMap and HashSet, O(C log C)
for TreeMap and TreeSet, O(C log C)

Note: cannot get this point if type of Map or Set is not clear or if illegal operation on Map or Set.
Note: if call to `maxVotes` inside loop, possible $O(C^2)$
Note: no points off for unnecessary constant (e.g. O(2C) instead of O(C))
Note: no points off for failure to convert from `Integer` to `int` and vice-versa.

# AP® Computer Science AB
# 2004 SCORING GUIDELINES

## Question 3

**Part A:** class declaration **2 pts**

**+1/2** `public class PredatorFish extends Fish`

**+1/2** `private` int instance variable (with appropriate identifier like `turnsNotEating`)

**+1** constructor
- **+1/2** attempt ( two of these three must be present:
  two parameters, call to super, initialize int instance variable in body)
- **+1/2** correct (int instance variable need not be set – defaults to zero)

**Part B:** `eat` **3 pts**

**+1/2** get location in front

**+1/2** get object from that location

**+1** test and remove locatable object from environment
- **+1/2** attempt (to remove an object that has a location, not the location itself)
- **+1/2** correct (call to `isEmpty` must be accompanied by call to `isValid` including `(Fish)` cast for `die`)

**+1** return correct boolean (in context of a test for eating)

Note: **-1** usage for missing declaration & initialization (using `env, theEnv` for `environment()`) and/or using private data when not accessible (`myLoc, myDir`)

**Part C:** `act` **4 pts**

**+1** update `turnsNotEating` (reset to zero or increment/decrement)
- **+1/2** attempt (one correct or both with reversed condition)
- **+1/2** correct

Note: can get the point if this is done correctly within eat method in Part B

**+1/2** correctly test `isInEnv()`

**+1/2** call `eat()`

**+1/2** correctly test `turnsNotEating`

**+1/2** call `die` correctly (in context of test for starvation)

**+1** call `super.act()`
- **+1/2** attempt (must have `super.act()`)
- **+1/2** correct (lose this for wrong placement)

# AP® Computer Science AB
# 2004 SCORING GUIDELINES

## Question 4

**Part A:** `peekMin` **3 pts**

**+1/2** start at root – root cannot be modified; if an iterative method is used, it must use local

**+1** traverse to leftmost node
- **+1/2** attempt – must arrive at left most node (on purpose) at some point in the code
- **+1/2** correct

**+1 1/2** return data (must get traverse attempt point to get any of the following points)
- **+1/2** attempt to return something from node, including node itself
- **+1/2** return `node.getValue()` or `node.getData()`
- **+1/2** correct (no deduction for missing downcast)

**Part B:** `addHelper` **6 pts**

**+1 1/2** base case
- **+1/2** check if `t == null`
- **+1** return new `TreeNode` with `new Item(obj)`
  - **+1/2** attempt (must create `new TreeNode` with some reference to `obj`)
  - **+1/2** correct

Note: If base case is not checked properly, or does not exist, and look-ahead check is done on `t.getLeft()` and `t.getRight()` then new node occurs within `setLeft` and `setRight`. Can get `new TreeNode` point if `t` is correctly returned.

**+4 1/2** recursive case
- **+1** get data from item at node
  - **+1/2** attempt (must have `t.getValue()` or `t.getData()`)
  - **+1/2** correct
- **+1** if equal, increment count
  - **+1/2** attempt – must show an equality test and use `incrementCount()`
  - **+1/2** correct
- **+1/2** traverse correct side – requires correct inequality test and correct direction selected
- **+1 1/2** recursive/repeated calls (eligible for these points even if traverse point is lost)
  - **+1/2** attempt
    - must attempt recursive calls to each side, or
    - some iteration involving each side
  - **+1** correct setting of each side with `setLeft()` and `setRight()`
- **+1/2** return `t` (separate from base case)
  - must return unchanged `t`
  - `return addHelper(…)` does not work

Note: After deduction on get data point, treat `t.getValue()` or `t.getData()` as correct data

# 2004 General Usage/Java

Most common usage errors are addressed specifically in rubrics with points deducted in a manner other than indicated on this sheet. The rubric takes precedence.

Usage points can only be deducted if the part where it occurs has earned credit.

A usage error that occurs once on a part when the same usage is correct two or more times can be regarded as an oversight and not penalized. If the usage error is the only instance, one of two, or occurs two or more times, then it should be penalized.

A particular usage error should be penalized only once in a problem, even if it occurs on different parts of a problem.

## Non-penalized Errors

case discrepancies

variable not declared when others are declared in some part of question

missing "new" for constructor call once, when others are present in question

default constructor called without parens for example, `new Fish;`

missing { } where indentation clearly conveys intent

`obj.method` instead of `obj.method()`

loop variables used outside loop

`[r,c]`, `(r)(c)` or `(r,c)` instead of `[r][c]`

= instead of `==` (and vice versa)

missing `( )` around `if/while` conditions

`length` - `size` confusion for array, `String`, and `ArrayList`, with or without `()`

missing downcast from collection or map

unnecessary construction of object whose reference is reassigned, for example

```
Direction dir = new Direction();
dir = f.Direction;
```

`private` qualifier on local variable

use "," instead of "+" for `String` in `System.out.print(str1, str2)`)

missing `;`s or missing `public`

extraneous code with no side-effect, for example a check for precondition

automatic conversion of `Integer` to `int` and vice-versa (this is legal in Java 1.5, called auto(un)boxing)

## Minor Errors (1/2 point)

misspelled/ confused identifier (e.g., `len` for `length` or `left()` for `getLeft()` )

no variables declared

`new` never used for constructor calls

`void` method returns a value

modifying a constant (`final`)

use `equals` or `compareTo` method on primitives, for example
`int x; …x.equals(val)`

use value 0 for `null`

use values 0, 1 for `false`, `true`

use of `itr.next()` more than once as same value within loop

use keyword as identifier

`[]` – `get` confusion

assignment dyslexia, for example,
`x + 3 = y;` for `y = x + 3;`

## Major Errors (1 point)

read new values for parameters or or instance variables
(prompts part of this point)

extraneous code which causes side-effect, for example, information written to output.

use interface or class name instead of variable identifier, for example `Simulation.step()` instead of `sim.step()`

`aMethod(obj)` instead of `obj.aMethod()`

use of object reference that is incorrect, for example, use of `f.move()` inside method of `Fish` class

use private data or method when not accessible

destruction of data structure (e.g. by using root reference to a `TreeNode` for traversal of the tree; this is often handled in the rubric)

*Note: Case discrepancies for identifiers fall under the "not penalized" category. However, if they result in another error, they must be penalized. Sometimes students bring this on themselves with their definition of variables. For example, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the one point deduction applies. Interpret writing to give benefit of the doubt to the student.*