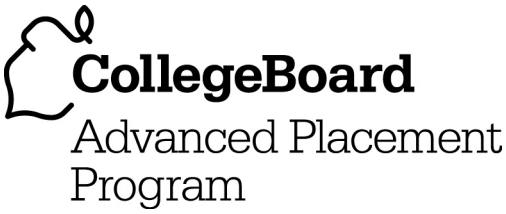

AP® Computer Science A
2004 Scoring Guidelines

AP® Computer Science A
2004 SCORING GUIDELINES

Question 1

Part A: `numWordsOfLength` **4 pts**

+1/2 declare and initialize count to zero (could be an empty list, length = 0)
(must show evidence that variable is used for counting or returned)

+1 loop over `myList`
- **+1/2** attempt (must reference `myList` in body)
- **+1/2** correct

+1/2 get `String` from `myList` (no deduction for missing downcast but local must be `String`)
(lose this if array syntax used)

+1 check length of `String`
- **+1/2** attempt (must be in context of loop)
- **+1/2** correct (array syntax is OK)

+1/2 increment count (must be within context of length check)
(lose this if count does not accumulate)

+1/2 return correct count (after loop is completed)

Part B: `removeWordsOfLength` **5 pts**

+2 loop over `myList`
- **+1** attempt (must reference `myList` in body)
- **+1** correct (must have attempt at removal, must not skip items)

+1 get `String` from `myList` (no deduction for missing downcast, but local must be `String`)
- **+1/2** attempt (must be in context of loop, array syntax is OK)
- **+1/2** correct (no array syntax)

+1 check length of `String`
- **+1/2** attempt (must be in context of loop)
- **+1/2** correct (array syntax is OK)

+1 remove
- **+1/2** attempt (must call `remove`, must refer to `myList` or an index of an element in `myList`)
- **+1/2** correct (no array syntax)

Usage:

-1/2 for `WordList` instead of `myList`

-1/2 for returning a value in part B

-1 for using `this` instead of `myList`, can lose in part A and again in part B (for max of -2)

AP® Computer Science A
2004 SCORING GUIDELINES

Question 2

Part A: `class Cat` **2 pts**

+1/2 `public class Cat extends Pet`

+1/2 Constructor correct (must call `super`)

+1 `speak` method
- **+1/2** attempt (method header matches abstract method, OK if `abstract` left in)
- **+1/2** correct

Part B: `class LoudDog` **3 pts**

+1/2 `public class LoudDog extends Dog`

+1 Constructor correct (must call `super`)

+1 1/2 `speak` method
- **+1** attempt (calls `super.speak()` *and* method header matches abstract method, OK if `abstract` left in)
- **+1/2** correct value returned

Part C: Kennel - `allSpeak` **4 pts**

+1 loop over `petList`
- **+1/2** attempt
- **+1/2** correct (must access `petList`)

+1 1/2 get pet from `petList` (no deduction for missing downcast from `petList`)
- **+1/2** attempt
- **+1** correct (local variable must be type `Pet`)

+1 1/2 print `p.getName()` and `p.speak()` for pet `p` (local variable not necessary)
- **+1/2** attempt (must have `xxx.getName()` or `xxx.speak()`, for some xxx)
- **+1** correct

Note: if done in-line with no local, no deduction for missing downcast.

Usage:
- **-1/2** `public` instance variable
- **-1** parent class name instead of `super`
- **-1/2** `getName` is overridden (other than `super.getName`) in part (a) and/or part (b)

(No deduction for other additional methods or constructors.)

AP® Computer Science A
2004 SCORING GUIDELINES

Question 3

Part A: `numUnder` **3 pts**

+1 1/2 calculate total fish needed in the pond
- **+1/2** attempt (must have calculation involving `minDensity` and either `numRows` or `numCols`)
- **+1** correct (must round up; `double` is OK if fixed later)

+1 1/2 return number of fish to add to the pond
- (a) zero when no fish should be added
- (b) number of fish to be added when positive
- **+1/2** attempt –must correctly return (a) or (b) (OK without conditional), **or** attempt to return (a) or (b) with condition involving `minDensity`
- **+1/2** correct (OK even when total fish needed in pond is incorrect)
- **+1/2** type returned is correct (`int`)

Part B: `randomLocation` **3 pts**

+1 get instance of `Random`
- **+1/2** attempt (`Math.random` or `Random r = new Random` gets this 1/2 point)
- **+1/2** correct (uses `RandNumGenerator.getInstance()`)

+1 generate random row and column in correct range
- **+1/2** attempt (must use some aspects of Random and env/loc)
- **+1/2** correct (must have instance of `Random` or correct use of `Math.random`) (lose this 1/2 if erroneous check for empty location is included)

+1 create and return location
- **+1/2** attempt (uses a `Location` object)
- **+1/2** correct

Part C: `addFish` **3 pts**

+1 loop until correct `numToAdd` have been added
- **+1/2** attempt
- **+1/2** correct

+1 generate random location until empty one is found
- **+1/2** attempt (must call `randomLocation` within loop for adding fish, must attempt to check if empty)
- **+1/2** correct

+1 create fish at this location (note: no local `Fish` variable is needed)
- **+1/2** attempt (`Fish(loc)` or `generateChild(loc)` gets this 1/2)
- **+1/2** correct (lose this 1/2 for adding twice, e.g., `theEnv.add(new Fish(…))`)

AP® Computer Science A
2004 SCORING GUIDELINES

Question 4

Part A: `forwardMoveBlocked` **1 pt**

+1 return boolean
- **+1/2** check a dir/pos pair
- **+1/2** correct

Part B: `move` **5 pts**

+1 check for item(s) on current tile and remove one
- **+1/2** attempt on current tile (might try to remove all items)
- **+1/2** correct

+1 1/2 check required conditions in context of attempt to move/turn
(body of each check must refer to `pos` or `facingRight`)
- **+1** separate check for empty tile (e.g., not in ELSE)
- **+1/2** check `forwardMoveBlocked`

+1 change direction (set direction to some value relative to current direction)
- **+1/2** toggle value
- **+1/2** if and only if originally blocked

+1 1/2 move (set position to value(s) relative to current position)
- **+1/2** attempt 2 directions (change position, not value at position)
- **+1/2** move 1 tile in proper direction
- **+1/2** if and only if originally not blocked

Part C: `clearHall` **3 pts**

+1/2 declare and initialize counter (must have some extra context relevant to counting)

+1 loop until done
- **+1/2** call to `hallIsClear` in loop
- **+1/2** correct

+1 robot action (in context of a loop)
- **+1/2** call `move`
- **+1/2** correctly determine number of times `move` is called

+1/2 always return number of times `move` is called (no credit for returning 0 with no call to `move` in code)

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.