AP® Computer Science A
2005 Scoring Guidelines

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,700 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three and a half million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

Visit the College Board on the Web: www.collegeboard.com.
AP Central is the official online home for the AP Program and Pre-AP: apcentral.collegeboard.com.

AP® COMPUTER SCIENCE A
2005 SCORING GUIDELINES

2005 A Question 1: Hotel Reservation

Part A: `requestRoom` **4 points**

+1 loop over `rooms`
- **+1/2** attempt (must reference multiple elements of `rooms` in body)
- **+1/2** correct

+1/2 test correct array entry for null (in context of loop)

+1 1/2 handle new reservation (in context of a loop)
- **+1/2** attempt to create new reservation (some sense of `Reservation` construction)
- **+1/2** correctly create reservation (if add to `rooms`, must be in `null` location & assignment correct)
- **+1/2** return reservation (only if `null` entry)

+1 handle wait list after loop or at appropriate time (only if full)
- **+1/2** add new guest to end of `waitlist` only once
- **+1/2** return `null`

Part B: `cancelAndReassign` **5 points**

+1 look up room number
- **+1/2** attempt (must call `res.getRoomNumber()` or use loop to find `res`)
- **+1/2** correct (must call `res.getRoomNumber()`)

+1/2 test `waitlist` to see if empty

+2 1/2 handle nonempty `waitList`
- **+1/2** get *first* entry from `waitList` (only if `waitlist` is not empty)
- **+1/2** create new `Reservation` } *can get these points by*
- **+1/2** assign `Reservation` to correct room } *correctly calling requestRoom*
- **+1/2** remove only first entry from `waitlist` (only if `waitlist` is not empty)
- **+1/2** return new `Reservation` (only if `waitlist` is not empty)

+1 handle empty case
- **+1/2** assign `null` to room (only if `waitList` is empty)
- **+1/2** return `null` (only if `waitList` is empty)

Note: If access using `get` on `rooms` is done more than once, deduct 1/2 usage point, not correctness (ditto for `set` on `rooms`).

AP® COMPUTER SCIENCE A
2005 SCORING GUIDELINES

2005 A Question 2: Ticket Sales

Part A: `Advance` **3 1/2 points**

+1/2 `class Advance extends Ticket` (no abstract)

+1/2 private data field (either days or price)

+1 1/2 constructor
- **+1/2** correct header
- **+1** correctly assign data field(s) (lose if reference to super's private data)

+1 `getPrice`
- **+1/2** correct header (must be `public` & `double`, no `abstract`, no parameters)
- **+1/2** return correct price

Part B: `StudentAdvance` **5 1/2 points**

+1/2 `class StudentAdvance extends Advance`

+1 1/2 constructor
- **+1/2** correct header
- **+1/2** attempt to call `super`
- **+1/2** correct call to `super`

+2 `getPrice`
- **+1/2** correct header (must be `public` & `double`, no `abstract`, no parameters)
- **+1** call `super.getPrice()`
- **+1/2** calculate and return correct price

+1 1/2 `toString`
- **+1/2** call `super.toString()`
- **+1** return `string` with correct phrase concatenated (lose this with a reference to super class's private data)

Usage: -1/2 in part A if `super()` appears in the constructor and it is not the first statement executed.

AP® COMPUTER SCIENCE A
2005 SCORING GUIDELINES

2005 A Question 3: ZigZag Fish

Part A: `nextLocation` **5 points**

+1/2 determine environment

+1/2 determine current location (lose this if reference inaccessible field)

+1/2 determine current direction (lose this if reference inaccessible field)

+2 determine diagonal locations
- **+1/2** attempt to access any neighbor of current location
- **+1/2** correctly access either forward-diagonal location
- **+1** access correct diagonal (based on `willZigRight`)

+1/2 check contents of diagonal location (`isEmpty`)

+1 return location (in some context of `willZigRight`)
- **+1/2** next location (only if empty)
- **+1/2** current location (only if blocked)

Part B: `move` **4 points**

+1/2 call `nextLocation()`

+1 check if no movement
- **+1/2** attempt
- **+1/2** correct

+1 reverse direction
- **+1/2** attempt
- **+1/2** correct (only if blocked, lose this if reference inaccessible field)

+1 1/2 move and update `willZigRight` (only if not blocked)
- **+1/2** `changeLocation(nextLoc)`
- **+1** correctly update `willZigRight`

AP® COMPUTER SCIENCE A
2005 SCORING GUIDELINES

2005 A Question 4: Improving Grades

Part A: `average` **3 points**

+1/2 initialize sum

+1 loop over `scores`
- **+1/2** attempt (must reference `scores` in body)
- **+1/2** correct (from first to last)

+1/2 add score to sum (in context of loop)

+1 calculate and return average
- **+1/2** attempt to calculate average
- **+1/2** return correct value
  (Check for `int` division; must be `double` quotient)

Part B: `hasImproved` **3 points**

+1 loop over `scores`
- **+1/2** attempt (must reference `scores` in body)
- **+1/2** correct (will lose this if index out of bounds)

+1 compare consecutive scores (in context of loop)
- **+1/2** attempt
- **+1/2** correct

+1 return correct `boolean`
- **+1/2** categorize entire array as improved or not improved
  (must be in context of comparing consecutive scores)
- **+1/2** correct value returned

Part C: `finalAverage` **3 points**

+1 call `hasImproved()`
- **+1/2** attempt
- **+1/2** correct

+1 return average of last half
- **+1/2** attempt to average half using `average`
- **+1/2** return correct average (only if improved)

+1 return average of all
- **+1/2** attempt to average all using `average`
- **+1/2** return correct average (only if not improved)

Note: Reimplementing code rather than calling available methods results in score of 0 for the portion of part C related to the code reimplementation.

2005 General Usage

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;`

`super(method())` instead of `super.method()`

formal parameter syntax (with type) in method call, e.g., `a = method(int x)`

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)

use class name in place of `super` either in constructor or in method call

Note: Case discrepancies for identifiers fall under the "not penalized" category. If two identifiers differ only in capitalization, the reader may use context to differentiate between them. For example, if a student declares "Fish fish;", then uses Fish.move() instead of fish.move(), the context allows for the reader to assume the object instead of the class. If context is not clear, say if the two identifiers have the same type, then a one point penalty must be applied.

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science A

Question 1

PART A:

```
public Reservation requestRoom(String guestName)
{
  for (int i = 0; i < rooms.length; i++)
  {
    if (rooms[i] == null)
    {
      rooms[i] = new Reservation(guestName, i);
      return rooms[i];
    }
  }
  waitList.add(guestName);
  return null;
}
```

PART B:

```
public Reservation cancelAndReassign(Reservation res)
{
  int roomNum = res.getRoomNumber();
  if (waitList.isEmpty())
  {
    rooms[roomNum] = null;
  }
  else
  {
    rooms[roomNum] = new Reservation((String)waitList.get(0), roomNum)
    waitlist.remove(0);
  }
  return rooms[roomNum];
}
```

alternate solution

```
public Reservation cancelAndReassign(Reservation res)
{
  int roomNum = res.getRoomNumber();
  rooms[roomNum] = null;
  if (!waitList.isEmpty())
  {
    requestRoom((String)waitlist.get(0));
    waitlist.remove(0);
  }
  return rooms[roomNum];
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science A

Question 2

PART A:

```
public class Advance extends Ticket
{
  private int daysInAdvance;

  public Advance(int numDays)
  {
    super();
    daysInAdvance = numDays;
  }

  public double getPrice()
  {
    if (daysInAdvance >= 10)
    {
      return 30.0;
    }
    else
    {
      return 40.0;
    }
  }
}
```

OR

```
public class Advance extends Ticket
{
  private double price;

  public Advance(int numDays)
  {
    super();
    if (numDays >= 10)
    {
      price = 30.0;
    }
    else
    {
      price = 40.0;
    }
  }

  public double getPrice()
  {
    return price;
  }
}
```

PART B:

```
public class StudentAdvance extends Advance
{
  public StudentAdvance(int numDays)
  {
    super(numDays);
  }

  public double getPrice()
  {
    return super.getPrice()/2;
  }

  public String toString()
  {
    return super.toString() + "\n(student ID required)";
  }
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science A

Question 3

PART A:

```
protected Location nextLocation()
{
  Environment env = environment();
  Location loc = location();
  Direction dir = direction();

  Location forward = env.getNeighbor(loc, dir);
  Location nextLoc;
  if (willZigRight)
  {
    nextLoc = env.getNeighbor(forward, dir.toRight());
  }
  else
  {
    nextLoc = env.getNeighbor(forward, dir.toLeft());
  }

  if (env.isEmpty(nextLoc))
  {
    return nextLoc;
  }
  else
  {
    return loc;
  }
}
```

PART B:

```
protected void move()
{
  Location nextLoc = nextLocation();
  if (nextLoc.equals(location())) {
    changeDirection(direction().reverse());
  }
  else
  {
    changeLocation(nextLoc);
    willZigRight = !willZigRight;
  }
}
```

Workshop Exam Materials
Canonical Solutions
2005 AP® Computer Science A

Question 4

PART A:

```
public double average(int first, int last)
{
  double sum = 0.0;
  for (int i = first; i <= last; i++)
  {
    sum += scores[i];
  }
    return sum/(last-first+1);
}
```

PART B:

```
public boolean hasImproved()
{
  for (int k = 0; k < scores.length-1; k++)
  {
    if (scores[k] > scores[k+1])
    {
      return false;
    }
  }
  return true;
}
```

PART C:

```
public double finalAverage()
{
  if (hasImproved())
  {
    return average(scores.length/2, scores.length-1);
  }
  else
  {
    return average(0, scores.length-1);
  }
}
```