2025

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free-Response Question 1

- ☑ **Scoring Guidelines**
- ☑ **Student Samples**
- ☑ **Scoring Commentary**

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â□□; a character that displays as ï¼□ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa

Question 1: Methods and Control Structures — 9 points

Canonical solution

a. **4 points**

```
public int walkDogs(int hour)
{
   int dogsToWalk = company.numAvailableDogs(hour);

   if (dogsToWalk > maxDogs)
   {
      dogsToWalk = maxDogs;
   }

   company.updateDogs(hour, dogsToWalk);
   return dogsToWalk;
}
```

b. **5 points**

```
public int dogWalkShift(int startHour, int endHour)
{
   int totalPay = 0;

   for (int hour = startHour; hour <= endHour; hour++)
   {
      int dogs = walkDogs(hour);
      int hourPay = 5 * dogs;
      if (dogs == maxDogs || (hour >= 9 && hour <= 17))
      {
         hourPay += 3;
      }
      totalPay += hourPay;
   }
   return totalPay;
}
```

a. `walkDogs`

	Scoring Criteria	Decision Rules	
1	Calls `DogWalkCompany` method(s) on `company`	Responses **can** still earn the point even if they • fail to save or use the returned value • call `numAvailableDogs` more than once • only call one of the methods Responses **will not** earn the point if they • use the incorrect parameter types • call either `numAvailableDogs` or `updateDogs` on nothing or on something other than `company`	**1 point**
2	Compares available dogs and `maxDogs` to determine number of dogs to walk	Responses **can** still earn the point even if they • call `numAvailableDogs` incorrectly • calculate an incorrect number of dogs that were walked	**1 point**
3	Calls `numAvailableDogs` with `hour` and `updateDogs` with `hour` and correct number of dogs to walk (*algorithm*)	Responses **can** still earn the point even if they • call either method on nothing or on something other than `company` Responses **will not** earn the point if they • calculate an incorrect number of dogs that were walked	**1 point**
4	Returns calculated value	Responses **can** still earn the point even if they • calculate an incorrect number of dogs that were walked • call `numAvailableDogs` multiple times Responses **will not** earn the point if they • return something other than an `int` • fail to return a value in some cases • print a value in addition to or instead of returning it	**1 point**

b. `dogWalkShift`

	Scoring Criteria	Decision Rules	
5	Loops from `startHour` to `endHour`, inclusive	Responses **can** still earn the point even if they • return from the loop early, as long as bounds would otherwise be correct	**1 point**
6	Calls `walkDogs` with `int` parameter (*in the context of a loop*)	Responses **can** still earn the point even if they • fail to save or use the returned value Responses **will not** earn the point if they • use an incorrect parameter type on any call to `walkDogs` • call the method on the class or on an object other than `this` (use of `this` is optional)	**1 point**
7	Calculates base pay for an hour	Responses **can** still earn the point even if they • fail to include the calculation in the loop • call `walkDogs` incorrectly	**1 point**
8	Calculates possible bonus for an hour's pay	Responses **can** still earn the point even if they • call `walkDogs` multiple times inside the loop • fail to include the calculation in the loop Responses **will not** earn the point if they • count the bonus twice when both conditions are true • count the bonus only when both conditions are true • count the bonus when it has not been earned • calculate bonus incorrectly	**1 point**
9	Accumulates total pay (*algorithm*)	Responses **can** still earn the point even if they • calculate base pay or bonus incorrectly • fail to return total pay (*return not assessed in this part*) Responses **will not** earn the point if they • fail to initialize variable used for total pay • call `walkDogs` multiple times inside the loop • do not accumulate base and bonus in the context of a loop • return from the loop early	**1 point**
	Question-specific penalties		
	None		

Part (a)

```
public int walkDogs(int hour){
   int doggies = company.numAvailableDogs(hour);
   if(doggies >= maxDogs){
      company.updateDogs(hour, maxDogs);
      return maxDogs;
   }
   company.updateDogs(hour, doggies);
   return doggies;
}
```

Part (b)

```
public int dogWalkShift(int startHour, int endHour){
   int totalSal = 0;
   for(int i = startHour; i <= endHour; i++){
      int dogs = walkDogs(i);
      totalSal += dogs * 5;
      if(dogs == maxDogs)
         totalSal += 3;
   }
   return totalSal;
}
```

Part (a)

```
public int walkDogs(int hour)
{
int i=0;
int dogs = numAvialableDogs(hour);
if(maxDogs >= dogs)
{
i = dogs;
}
else
{
i = maxDogs;
}
updateDogs(hour,i);
return i;
}
```

Part (b)

```
public int dogWalkShift(int startHour, int endHour)
{
int total = 0;
for ( int i = startHour ; i <= endHour;i++)
{
if( this.walkDogs() == maxDogs || (startHour >9 && endHour<17)
{
total += (this.walkDogs() * 5 +3)
}
else
{
toal += (this.walkDogs() * 5)
}
}
}
```

Part (a)

```
public int walkDogs(int hour)
{
   int dogsWalked = 0;

   if (numAvailableDogs(hour) > 0)
   {
     dogsWalked += numAvailableDogs() - maxDogs;
     updateDogs(hour, numberDogsWalked);
   }

   return dogsWalked;
}
```

Part (b)

```
public int dogWalkShift(int startHour, int endHour)
{
  int amountEarned = 0;

  for (int i = startHour, i <= endHour; i++)
  {
    walkDogs(i);

    amountEarned += 5;

    if (maxDogs || (startHour <= 9 && endHour >= 17))
    {
      amountEarned += 3;
    }
  }

  return amountEarned;

}
```

Question 1

Note: Student samples are quoted verbatim and may contain spelling and grammatical errors.

Overview

NEW for 2025: The question overviews can be found in the *Chief Reader Report on Student Responses* on AP Central.

Sample: 1A
Score: 8

In part (a) point 1 was earned because both `numAvailableDogs` and `updateDogs` are called on the `company` instance variable. Point 2 was earned because the conditional statement compares the number of available dogs to `maxDogs`. This point is earned even if an incorrect number of dogs is passed into `updateDogs` based on the comparison `if(doggies >= maxDogs)`. The focus of this point is the comparison of a number of dogs to `maxDogs` to determine the number of dogs to walk. Point 3 was earned because the correct parameters are used for both `numAvailableDogs` and `updateDogs` and the correct number of dogs are walked in the call to `updateDogs`. Point 4 was earned because a calculated integer is returned with `int doggies = company.numAvailableDogs(hour)` and `return doggies`.

In part (b) point 5 was earned because the loop iterates over all hours from `startHour` to `endHour`, inclusive. Point 6 was earned because the call to `walkDogs(i)` has the correct parameter type and is called within the loop. Point 7 was earned because the base pay for at least one hour is calculated with `int dogs = walkDogs(i)` and `totalSal += dogs * 5`. The calculation does not need to occur inside the loop to earn this point. Point 8 was not earned because there is no condition to check if the current hour is a peak hour. As noted in the prompt, a peak hour is an hour from `9` to `17`, inclusive. Even though the response checks to see if the maximum number of dogs were walked in the hour, both conditions must be tested to earn this point. Point 9 was earned because the algorithm initializes a variable to represent the total pay and accumulates both base pay and bonus calculations in the loop. The calculations for both base pay and bonus pay don't need to be correct to earn Point 9 since those are assessed in Point 7 and Point 8.

Sample: 1B
Score: 5

In part (a) point 1 was not earned because both `numAvailableDogs` and `updateDogs` are not called on the `company` instance variable. Both methods must be called on the `company` object each time to earn the point. Point 2 was earned because the conditional statement compares the number of available dogs to `maxDogs`. Point 3 was earned because the correct parameters are used for both `numAvailableDogs` and `updateDogs` and the correct number of dogs to walk is calculated. Note that the spelling error (`numAvialableDogs`) is a no penalty error because there is no ambiguity, as noted on the *Applying the Scoring Criteria* page of the Scoring Guidelines. This response does not indent the contents of each block. Indentation is not required in Java, and responses without indentation are not penalized. However, without indentation, the curly braces are the only means to determine where blocks of code begin and end. Point 4 was earned because a calculated integer is returned.

Question 1 (continued)

In part (b) point 5 was earned because the loop iterates over all hours from `startHour` to `endHour`, inclusive. Point 6 was not earned because the `walkDogs` method call requires an `int` parameter. Point 7 was earned because the base pay for at least one hour is calculated. Note that the variable `total` is misspelled. This is a nonpenalized error since there is no ambiguity. Point 8 was not earned because the comparison `(startHour >9 && endHour<17)` does not refer to the hour being considered, `i`. Additionally, a correct comparison would include the endpoints, as in `i>= 9 && i<= 17`. Note that the keyword `this` is optional for either the call to `walkDogs` or to access the private instance variable `maxDogs`. The keyword `this` refers to the object that has called the method in which it is used. Here it explicitly refers to the object that originally called the `dogWalkShift` method; without it the method would implicitly be called by that object. Point 9 was not earned because the response calls `this.walkDogs()` multiple times in the loop. The method `walkDogs` calls the `updateDogs` method, removing too many dogs from the pool of available dogs.

Sample: 1C
Score: 4

In part (a) point 1 was not earned because both `numAvailableDogs` and `updateDogs` are not called on the `company` instance variable. Both methods must be called on the `company` object every time to earn this point. Point 2 was not earned because the conditional expression `(numAvailableDogs(hour) > 0)` compares available dogs to `0` instead of `maxDogs`. Point 3 was not earned because the second call to `numAvailableDogs()` is missing the parameter. All calls to `dogWalkCompany` methods must have the correct parameters to earn this point. Point 4 was earned with the `return dogsWalked;` statement, since this returns a calculated `int` value.

In part (b) point 5 was earned because the loop iterates from `startHour` to `endHour`, inclusive. The comma instead of the semicolon in the `for` loop header is a no penalty error, as noted on page 2 of the *Applying the Scoring Criteria* page of the Scoring Guidelines. Point 6 was earned because `walkDogs(i)` correctly retrieves the number of dogs walked within a loop. Point 7 was not earned because the base pay is increased with `amountEarned += 5`, which is adding a constant and not a calculated value of the number of dogs walked in an hour. Point 8 was not earned because `maxDogs` is an `int` value and must be compared to another numerical value in a conditional statement. It should be compared to the number of dogs walked in an hour. Additionally, the comparison `(startHour <= 9 && endHour >= 17)` does not refer to the hour being considered, `i`. Furthermore, the operators are backwards and should be `i >= 9 && i <= 17`. Point 9 was earned because the total pay is initialized, and the algorithm accumulates base pay and bonus calculations in the loop.