2023

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.*

Question 1: Methods and Control Structures — 9 points

Canonical solution

(a) — **5 points**

```
public int findFreeBlock(int period, int duration)
{
   int blockLength = 0;

   for (int minute = 0; minute < 60; minute++)
   {
      if (isMinuteFree(period, minute))
      {
         blockLength++;
         if (blockLength == duration)
         {
            return minute - blockLength + 1;
         }
      }
      else
      {
         blockLength = 0;
      }
   }
   return -1;
}
```

(b) — **4 points**

```
public boolean makeAppointment(int startPeriod,
                               int endPeriod,
                               int duration)
{
   for (int period = startPeriod;
        period <= endPeriod;
        period++)
   {
      int minute = findFreeBlock(period, duration);
      if (minute != -1)
      {
         reserveBlock(period, minute, duration);
         return true;
      }
   }
   return false;
}
```

(a) `findFreeBlock`

	Scoring Criteria	Decision Rules	
1	Loops over necessary minutes in an hour	Responses **can** still earn the point even if they • loop over fewer than 60 minutes as long as at least (60 – duration + 1) minutes are included • loop over 60 minutes and use a `boolean` to indicate that a free block has been found	**1 point**
2	Calls `isMinuteFree` with `period` and another `int` parameter	Responses **can** still earn the point even if they • call `isMinuteFree` with invalid parameters due to incorrect loop bounds Responses **will not** earn the point if they • use incorrect parameter types • order the parameters incorrectly • call the method on the class or on an object other than `this` (use of `this` is optional)	**1 point**
3	Keeps track of contiguous free minutes in a block (*algorithm*)	Responses **can** still earn the point even if they • call `isMinuteFree` incorrectly Responses **will not** earn the point if they • fail to reset when a nonfree minute is found • call `isMinuteFree` with minutes `>= 60`	**1 point**
4	Checks whether a valid block of `duration` minutes has been found	Responses **can** still earn the point even if they • maintain a `boolean` instead of accumulating the block length	**1 point**
5	Calculates and returns starting minute and `-1` appropriately based on identified block (*algorithm*)	Responses **will not** earn the point if they • are off by one on the returned value	**1 point**
		Total for part (a)	**5 points**

(b) `makeAppointment`

	Scoring Criteria	Decision Rules	
6	Loops over periods from `startPeriod` through `endPeriod` (*no bounds errors*)		**1 point**
7	Calls `findFreeBlock` and `reserveBlock` with correct number of `int` parameters, representing a period and minute as appropriate, and `duration`	Responses **can** still earn the point even if they • use incorrect parameter values Responses **will not** earn the point if they • use incorrect parameter types • order the parameters incorrectly • call the methods on the class or on an object other than `this` (use of `this` is optional)	**1 point**
8	Guards call to method to reserve a block by determining that starting minute is not `-1`		**1 point**
9	Books correct appointment and returns appropriate `boolean` (*algorithm*)	Responses **can** still earn the point even if they • have incorrect bounds in the loop • call `findFreeBlock` or `reserveBlock` incorrectly Responses **will not** earn the point if they • fail to return `true` or `false` • return before the call to `reserveBlock`	**1 point**
			4 points
	Question-specific penalties		
	None		
		Total for question 1	**9 points**

Alternate Canonical for Part (a)

```
public int findFreeBlock(int period, int duration)
{
   for (int startMin = 0; startMin < 60 - duration + 1; startMin++)
   {
      boolean isBlockFree = true;
      for (int min = 0; min < duration; min++)
      {
         if (!isMinuteFree(period, min + startMin))
         {
            isBlockFree = false;
         }
      }
      if (isBlockFree)
      {
         return startMin;
      }
   }
   return -1;
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ● Question 2 ○ Question 3 ○ Question 4 ○

Begin your response to each question at the top of a new page.

a)
```
public int findFreeBlock (int period, int duration) {
    int free = -1;
    for (int i = 0 ; i < 60 - duration + 1 ; i++) {
        int count = 0;
        for (int j = i ; j < i + duration ; j++) {
            if (isMinuteFree (period, j)) {
                count ++;
            }
            if (count == duration) {
                free = i ;
            }
        }
    }
    return free;
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ● Question 2 ○ Question 3 ○ Question 4

Begin your response to each question at the top of a new page.

b)
```
public boolean makeAppointment (int startPeriod, int endPeriod,
                                int duration) {
    int s = 0;
    for (int p = startPeriod; p < endPeriod + 1; p++) {
        if (findFreeBlock (p, duration) != -1) {
            s = findFreeBlock (p, duration);
            reserveBlock (p, s, duration);
            return true;
        } else {
            return false;
        }
    }
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ● Question 2 ○ Question 3 ○ Question 4 ○

Begin your response to each question at the top of a new page.

a)
```
Public int findFreeBlock(int period, int duration) {
    boolean open;
    for(int i=0; i<=59; i++) {
        if (isMinuteFree(period, i)) {
            open = true;
            return i;
        }else{
            open = false;
            return -1;
        }
    }
}
```

b)
```
Public boolean makeAppointment (int startPeriod, int endPeriod,
                                int duration){
    for (int i = startPeriod; i<= endPeriod; i++) {
        if (isMinuteFree(i, findFreeBlock(i, duration))) {
            reserveBlock(i, findFreeBlock(i, duration), duration);
            return true;
        }else{
            return false;
        }
    }
}
```

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1	Question 2	Question 3	Question 4
●	○	○	○

Begin your response to each question at the top of a new page.

a.)

```
Public int findFreeBlock(int period, int duration){
while( 1 <=period <=8){
    while(1<=duration <=60){
int count=0;
  while(i=0 ; i <= duration; i++){
     if( isMinuteFree(int period, int i) == true){
        count++;
        }
     if(count ==duration){
     return indexOf(i)
        }
    }
  }
}
```


Use a pencil only. Do NOT write your name. Do NOT write outside the box.

Important: Completely fill in the circle that corresponds to the question you are answering on this page.

Question 1 ○ Question 2 ○ Question 3 ○ Question 4 ○

Begin your response to each question at the top of a new page.

D.)
```
public boolean makeAppointment(int startPeriod,
int endPeriod, int duration){
while(startperiod <= endperiod){
    int startMinute = 0;
    if(reverseBlock(int startperiod, int startMinute,
    int duration) != -1){


}

int indexof(findFreeBlock(int startperiod, int duration)
    = startMinute;
}
Startperiod++;
}
```

Question 1

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

Overview

This question tested the student's ability to:

- Write program code to create objects of a class and call methods.
- Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.

More specifically, this question assessed the ability to iterate through a range, determine when a proper stopping condition has been met before the end of the range, call instance methods from other instance methods of the same class, and use a method's return value in a conditional expression.

In part (a) students were asked to find the first available block of a specified duration within a specified hour, using the class (static) method `isMinuteFree` to determine availability. To do this, they had to initialize a variable to track whether a block of the requested length could fit in the block of available minutes starting at each candidate starting minute. They had to iterate through all necessary minutes in the hour, call the given method on each necessary minute, and conditionally update the tracking variable—either to continue tracking the candidate block or to reset for the next candidate block, as appropriate. After the loop, they had to return `-1` if no block of `duration` or more minutes was available. There were essentially two interacting algorithms in part (a): keeping track of contiguous blocks of free minutes and choosing the correct starting minute (or `-1`).

In part (b) students were asked to reserve the first available block of a specified duration within a specified set of hours. They were instructed to use the method written in part (a), `findFreeBlock`, to determine the appropriate minute to book and to use another method from the class's API, `reserveBlock`, to book the appointment, then return `true`. They had to demonstrate the ability to determine if a reservation could *not* be made, in which case the method needed to return `false` after the loop.

Sample: 1A
Score: 7

In part (a) point 1 was earned because the outer loop header is written to consider all necessary minutes in an hour, iterating from `0` up to but not including `60 - duration + 1`. Point 2 was earned by using the correct syntax to call the method `isMinuteFree` with arguments `period` and `j`. Point 3 was earned using a nested loop solution. The response uses an inner loop to determine if there is a block of `duration` contiguous minutes. The variable `count` is initialized to `0` at the beginning of each potential block of `duration` contiguous minutes (prior to the start of the inner loop), then incremented within the inner loop for each successive free minute. The inner loop executes up to but not including `duration` times. In the body of the inner loop, the argument `j` in the `isMinuteFree` method call is always a valid minute, as required to earn this point.

Question 1 (continued)

Note that responses that call `isMinuteFree` with a value greater than or equal to `60` will not earn this point. Point 4 was earned by determining whether a block of `duration` minutes has been found, using the `count == duration` check in the body of the inner loop. Point 5 was not earned because the starting minute of the first available block is not always returned. When a block of length `duration` is found, `i` is not returned immediately, so a different block could be found in a subsequent iteration of the outer loop.

In part (b) point 6 was earned because the loop header is written to iterate from `startPeriod` through `endPeriod`, inclusive. The loop goes up to, but does not include, `endPeriod + 1`, and therefore, the last value assigned to the loop variable `p` is `endPeriod`. Note that the response returns in both the `if` and the `else` blocks, based on the result returned by `findFreeBlock`. As a result, the loop iterates only one time. The early return from the loop is assessed in point 9 and is not assessed here. Point 7 was earned because the response includes correct calls to both `findFreeBlock` and `reserveBlock` with correct arguments. The call to `findFreeBlock` includes two `int` arguments, where the first argument, `p`, represents the period, and the second argument is `duration`. The call to `reserveBlock` includes three `int` arguments. The first argument, `p`, represents the period. The second argument, `s`, represents the starting minute of an identified free block in period `p`. The third argument is `duration`. Point 8 was earned by guarding the `reserveBlock` call with a condition that checks for a return value from `findFreeBlock` that is not `-1`. The fact that `findFreeBlock` is called twice is OK because both calls return the same value. Point 9 was not earned because of the early return inside the loop. Because there is a return in both blocks of the conditional statement, the loop executes only once. As a result, the response may not book the correct appointment or return the appropriate `boolean` value.

Sample: 1B
Score: 4

In part (a) point 1 was earned because the loop header is written to iterate over the necessary minutes in an hour by iterating from 0 to 59, inclusive. Note that the response includes a `return` in both the `if` and the `else` blocks. As a result, the loop iterates only one time. The early return is assessed in point 5 and is not assessed here. Also, the use of the common mathematical symbol `≤` for `<=` is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" section on page 1 of the Scoring Guidelines for a complete list.) Point 2 was earned by use of correct syntax to call the method `isMinuteFree` with arguments `period` and an `int` variable, `i`. Point 3 was not earned because the response does not keep track of contiguous free minutes in a block. The response would have to identify contiguous free minutes and use a variable to track them. The loop includes an early return that is assessed in point 5, but even without the early return, the variable `open` is updated for every minute `i` without considering its previous value. Point 4 was not earned for either of the following two reasons. First, the response does not identify a valid block of contiguous free minutes. To identify such a block, a response must use a variable (typically an `int` or `boolean`) to consider the combined states of at least two distinct free minutes. This can be done using an accumulator to store the current number of contiguous free minutes identified or by using a `boolean` to identify a valid contiguous block. Second, there is no

Question 1 (continued)

comparison involving `duration` to determine if a block of length `duration` has been found. Point 5 was not earned for either of the following two reasons. First, the response does not identify a block of contiguous free minutes. Second, the response includes an early return inside the loop. Because there is a return in both blocks of the conditional statement, the loop executes only once. As a result, the response may not return the appropriate value.

In part (b) point 6 was earned because the loop header is written to iterate from `startPeriod` through `endPeriod`, inclusive. Note that the response includes a `return` in both the `if` and the `else` blocks. As a result, the loop iterates only one time. The early return is assessed in point 9 and is not assessed here. Point 7 was earned because the response includes a correct call to both `findFreeBlock` and `reserveBlock` with correct arguments. The call to `findFreeBlock` includes two `int` arguments, where the first argument, `i`, represents the period, and the second argument is `duration`. The call to `reserveBlock` includes three `int` arguments. The first argument, `i`, represents the period. The second argument is the value returned by a correct call to `findFreeBlock`, which represents the starting minute of an identified free block. The third argument is `duration`. Point 8 was not earned because the guard does not compare a starting minute with `-1`. Point 9 was not earned because of the early return inside the loop. Because there is a return in both blocks of the conditional statement, the loop executes only once. As a result, the response may not book the correct appointment or return the appropriate `boolean` value.

Sample: 1C
Score: 2

In part (a) point 1 was not earned because none of the loops are syntactically valid. The first two loops use chained relational operators, which are not valid Java syntax. The third loop is a `while` loop that uses `for` loop syntax. Point 2 was not earned because the call to `isMinuteFree` includes types for the arguments; this is incorrect method call syntax. Point 3 was not earned because the response does not reset the `count` variable when an unavailable minute is found. As a result, `count` continues to be incremented even after a block of length `duration` has been encountered. Point 4 was earned by determining whether a block of `duration` minutes has been found, using the `count == duration` check in the inner loop. Note that point 4 can still be earned even though `count` is not reset to `0` when an unavailable minute is found. Point 5 was not earned for either of the following reasons. First, `-1` is never returned to indicate that an appropriate block was not found. Second, when an appropriate block has been identified, the response returns `indexOf(i)`, which cannot be interpreted as the starting minute of the block.

In part (b) point 6 was earned because the loop header is written to iterate from `startPeriod` through `endPeriod`, inclusive. Point 7 was not earned because the calls to `findFreeBlock` and `reverseBlock` include types for the arguments; this is incorrect method call syntax. Note that if the method calls had been syntactically valid, `reverseBlock` would be considered a spelling discrepancy where there is no ambiguity. This is one of the minor errors for which no penalty is assessed on this exam. (See the "No Penalty" section on page 1 of the Scoring Guidelines for a complete list.) Point 8 was not earned because the response compares the return value of `reserveBlock` to `-1` instead of comparing the return value of `findFreeBlock` to `-1`, and the

Question 1 (continued)

comparison is not used to guard a call to `reserveBlock`. Point 9 was not earned because the response does not return a value.