2019

AP® CollegeBoard

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free Response Question 1

☑ Scoring Guideline

☑ Student Samples

☑ Scoring Commentary

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Apply the question assessment rubric 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.*

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Question 1: Calendar

Part (a)	`numberOfLeapYears`	**5 points**

Intent: *Return the number of leap years in a range*

+1 Initializes a numeric variable

+1 Loops through each necessary year in the range

+1 Calls `isLeapYear` on some valid year in the range

+1 Updates count based on result of calling `isLeapYear`

+1 Returns count of leap years

Part (b)	`dayOfWeek`	**4 points**

Intent: *Return an integer representing the day of the week for a given date*

+1 Calls `firstDayOfYear`

+1 Calls `dayOfYear`

+1 Calculates the value representing the day of the week

+1 Returns the calculated value

Question-Specific Penalties

-1 (t) Static methods called with `this`.

Question 1: Scoring Notes

Part (a) `numberOfLeapYears`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Initializes a numeric variable		• use the variable for loop control only
+1	Loops through each necessary year in the range		• consider years outside the range
+1	Calls `isLeapYear` on some valid year in the range	• do not use a loop	
+1	Updates count based on result of calling `isLeapYear`	• do not use a loop • do not initialize the counter	• use result as a non-`boolean`
+1	Returns count of leap years	• loop from `year1` to `year2` incorrectly • do not initialize the counter	• do not use a loop • update or initialize the counter incorrectly • return early inside the loop
Part (b) `dayOfWeek`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Calls `firstDayOfYear`		• do not use the given year
+1	Calls `dayOfYear`		• have arguments out of order
+1	Calculates the value representing the day of the week		• make any error in the calculation
+1	Returns the calculated value	• return the value from calling `firstDayOfYear` or `dayOfYear`	• return a constant value

Question 1: Calendar

Part (a)

```
public static int numberOfLeapYears(int year1, int year2)
{
   int count = 0;
   for (int y = year1; y <= year2; y++)
   {
      if (isLeapYear(y))
      {
         count++;
      }
   }
   return count;
}
```

Part (b)

```
public static int dayOfWeek(int month, int day, int year)
{
   int startDay = firstDayOfYear(year);
   int nthDay = dayOfYear(month, day, year);
   int returnDay = (startDay + nthDay - 1) % 7;
   return returnDay;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

1Aa

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
{
   int numLeap = 0;
   while(year1 <= year2) {
      if (isLeapYear(year1)) {
          numLeap ++;
      }
      year1 ++;
   }
   return numLeap;
}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

1Ab

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
{
    int dayYear = dayOfYear(month, day, year);
    int FirstDay = FirstDayOfYear(year);
    int value=(dayYear % 7) + FirstDay;
    if(value > 7){
        return (Value -7 -1);
    }
    else {
        return value;
    }
}
```

GO ON TO THE NEXT PAGE.

1Ba

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
    int cntr=0;
    for(int y = year1; y <= year2; y++) {
       if( y %4 == 0) {
          cntr += 1; }}
     return cntr; }
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

1Bb

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year) {
    int firstDay = firstDayOfYear(year);
    int whichDay = dayOfYear(month, day, year);
    boolean leap = isLeapYear(year);
    int answer;
    if (leap == false) {
        int remainder = whichDay % 7;
        answer = remainder + firstDay - 1;
        if (answer > 6) {
            answer = answer - 7; } }
    else {
        int remainder = whichDay % 7;
        answer = remainder + firstDay;
        if (answer > 6) {
            answer = answer - 7; } }
    return answer; }
```

GO ON TO THE NEXT PAGE.

1Ca

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
int numLeaps =0;
for(int x = year1+1 ; x<year2;x++){
if ( x. isLeap Year (x) == true)
 numLeaps ++;
}
 return numLeaps;
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

1Cb

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

```
int dayOfTheWeek = 0;
int daysOfYear = dayOfWeek.dayOfYear(month, day, year);
```

GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2019 SCORING COMMENTARY

Question 1

Overview

This question tested the student's ability to:

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

More specifically, this question assessed the ability to use numeric primitive types, iterate through a range, call static methods, and use a method's return value in a conditional expression.

In part (a) students were asked to count the number of years within a given range (inclusive) that were leap years. They were provided the method `isLeapYear` to determine whether a particular year was a leap year and were instructed to call this method rather than implementing the (unspecified) leap year criteria. They were expected to initialize a numeric counter, iterate through all years in the range, call the given method on each year, use the result to conditionally update the counter, and return the counter after the iteration.

In part (b) students were asked to determine the day of the week on which a given date (month, day, and year) falls. They were provided the method `firstDayOfYear` to determine the day of the week of the first day of a year, encoded 0 through 6. They were also provided the method `dayOfYear` to determine the ordinal date (the number of days of the year that have elapsed, including the given day), from 1 through 366. The students were instructed to call these methods rather than implementing the (unspecified) logic to compute them.

Sample: 1A
Score: 8

In part (a) the response begins with a line of code that declares an `int` variable `numLeap`. The response initializes the numeric variable by assigning it the value `0` and earned point 1. Point 2 was earned because the loop is set up in such a way that `year1` represents all the years between the original value of `year1` and `year2`, inclusive. The parameter, `year1`, can be changed without destroying persistent data because `int` is a primitive type. Point 3 was earned for the correct call to the `isLeapYear` method. Because `numLeap++` is executed conditionally in the context of calling `isLeapYear`, point 4 was earned. After the loop is finished, the last statement returns the accumulated count of leap years, and point 5 was earned. Part (a) earned 5 points.

In part (b) the response earned point 6 for the correct call to the `firstDayOfYear` method and point 7 for the correct call to the `dayOfYear` method. After the addition of the result of the remainder calculation and `firstDay`, `7` is a possible value of the variable `value`. This would cause the conditional expression to evaluate to `false` and the `else` clause to be executed. The `else` clause is missing the `-1` to adjust for `dayOfYear` returning a count beginning with one, which violates the method specification requiring a return value between `0` and `6`, inclusive. As a result, point 8 was not earned. Point 9 was earned because `value` is calculated, although possibly incorrectly, and is returned by all paths. Part (b) earned 3 points.

Sample: 1B
Score: 6

In part (a) point 1 was earned for the initialization of `cntr`. The loop structure is correct and earned point 2. There is no call to the `isLeapYear` method, so point 3 was not earned. Point 4 was not earned because the update of `cntr` was not based on the result of a call to `isLeapYear`. Even though points 3 and 4 were not earned, the code demonstrates a conditional counting of years. The `return` statement appears after the loop, returning the counter variable, so point 5 was earned. Part (a) earned 3 points.

In part (b) a correct call to `firstDayOfYear` earned point 6. The call to `dayOfYear` includes the three required arguments in the correct order and earned point 7. The call to `isLeapYear` is not necessary because the `dayOfYear` method already accounts for leap years. The response includes code for two separate cases, and point 8 was not earned because the `-1` adjustment is only applied in non-leap years. Point 9 was earned because `answer`, a calculated value, is returned in all cases. Part (b) earned 3 points.

Sample: 1C
Score: 3

In part (a) the numeric variable `numLeaps` is initialized, and point 1 was earned. The loop structure includes the range between `year1` and `year2`, exclusive. Because the method specifies that the range of years is between `year1` and `year2`, inclusive, point 2 was not earned. Point 3 was not earned because the call to `isLeapYear` is incorrect. As a `static` method, `isLeapYear` cannot be invoked as though it were an instance method and `x` is declared as an `int`—primitive types have no instance methods. Although the method call is syntactically incorrect, point 4 was earned because the update is based on the result of calling `isLeapYear`. The `return` statement earned point 5, even though the years may be undercounted due to the loop bounds error. Part (a) earned 3 points.

In part (b) point 6 was not earned because there is no call to the `firstDayOfYear` method. The `dayOfYear` method call is syntactically incorrect because it is preceded by `dayOfWeek` and did not earn point 7. Points 8 and 9 were not earned because there is no calculation and no `return` statement. Part (b) earned no points.