2022

AP CollegeBoard

AP® Computer Science A

Sample Student Responses and Scoring Commentary

Inside:

Free-Response Question 2

- ☑ **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 2: Class — 9 points

Canonical solution

9 points

```
public class Textbook extends Book
{
   private int edition;

   public Textbook(String tbTitle, double tbPrice,
                   int tbEdition)
   {
      super(tbTitle, tbPrice);
      edition = tbEdition;
   }

   public int getEdition()
   {
      return edition;
   }

   public boolean canSubstituteFor(Textbook other)
   {
      return other.getTitle().equals(getTitle()) &&
             edition >= other.getEdition();
   }

   public String getBookInfo()
   {
      return super.getBookInfo() + "-" + edition;
   }
}
```

`Textbook`

	Scoring Criteria	Decision Rules	
1	Declares class header (must not be `private`): `class Textbook extends Book`		**1 point**
2	Declares constructor header: `public Textbook(String ___, double ___, int ___)`		**1 point**
3	Constructor calls `super` as the first line with the appropriate parameters		**1 point**
4	Declares appropriate `private` instance variable and uses appropriate parameter to initialize it	Responses **will not** earn the point if they • omit the keyword `private` • declare the variable outside the class, or in the class within a method or constructor • redeclare and use the instance variables of the superclass	**1 point**

5	Declares at least one required method and all declared headers are correct:
	`public boolean canSubstituteFor(Textbook ___)`
	`public int getEdition()`
	`public String getBookInfo()`
		Responses **will not** earn the point if they
		• exclude `public`
			1 point

6	`getEdition` returns value of instance variable	Responses **will not** earn the point if they • fail to create an instance variable for the edition	**1 point**
7	`canSubstituteFor` determines whether `true` or `false` should be returned based on comparison of book titles and editions (*algorithm*)	Responses **can** still earn the point even if they • fail to return (*return is not assessed for this method*) • access the edition without calling `getEdition` • redeclare and use the `title` variable of the superclass instead of calling `getTitle` Responses **will not** earn the point if they • fail to use `equals` • call `getTitle` incorrectly in either case	**1 point**
8	`getBookInfo` calls `super.getBookInfo`	Responses **can** still earn the point even if they • redeclare and use the instance variables of the superclass Responses **will not** earn the point if they • include parameters	**1 point**

9	Constructs information string	Responses **can** still earn the point even if they • call `super.getBookInfo` incorrectly • fail to call `super.getBookInfo` and access `title` and `price` directly • fail to return (*return is not assessed for this method*) Responses **will not** earn the point if they • omit the literal hyphen(s) in the constructed string • omit the edition in the constructed string • concatenate strings incorrectly	**1 point**

Question-specific penalties
None

Total for question 2 9 points

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.

```
public class Textbook extends Book{
        private      int edition;

    public Textbook(String ti, double pr, int ed){
        super(ti, pr);
        edition = ed;
    }
    public String getBookInfo(){
        return  super.getBookInfo() + "-" + edition;
    }
    public boolean canSubstituteFor(Textbook book){
        if( this.getTitle().equals(book.getTitle()) && this.getEdition() == book.getEdition())
            returns true;
        else
            returns false;

    public  int getEdition(){
        return edition;
    }
```

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.

```
public Class Textbook extends Book {
    private String t;
    private double p;
    private int e;

    public Textbook (String ti, double pr, int ed) {
        super(ti, pr);
        e = ed;
    }
    public int getEdition(){
        return e;
    }
    public String getBookInfo(){
        return t + "-" + p + "-" + e;
    }
    public boolean canSubstituteFor(Textbook s){
                                    → boolean canSub = false;
        if(this.t.equals(s.t)){
            canSub = true;
            if(this.getEdition().equals(s.getEdition()){
                canSub = true;
            }
            else {
                canSub = false;
            }
        }
        return canSub;
    }
}
```


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.

```
Public class Textbook extends Book{
private int edition;
Public String getBookInfo(){
  return title+"-"+price+"-"+edition;}
Public boolean canSubStituteFor(String title2)
{ if(title.substring(0,(title.length()-4)) == title2.substring(0,(title.length()-4))
&& title.substring((title.length()-4)) = title2.substring(title2.length()-4))
    return true;}
```

Question 2

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 define a new type by creating a class.
- Write program code to call methods.
- Write program code to satisfy method specifications using expressions and conditional statements.

Students were given a class, `Book`, and asked to design a subclass called `Textbook`. The `Book` class contained the title and the price of the book and accessor methods for this information. In implementing a solution, students were expected to demonstrate an understanding of class constructor and method header syntax. Students were expected to properly declare and initialize a new `private` instance variable to maintain the edition number. Further, students had to recognize that the `title` and `price` variables in the `Book` class were `private` and could not be accessed directly by the code. Students had to utilize the mechanisms of inheritance to initialize and access these variables by way of a `super` constructor call from the `Textbook` class to the `Book` class.

The specification for the `Textbook` class required two new methods be added: `getEdition` to return the edition of the textbook and `canSubstituteFor` to check if the textbook could be substituted for a given textbook. In the `canSubstituteFor` method, students were expected to compare the titles of two `Textbook` objects via the `equals` method of the `String` class. Students were also expected to use arithmetic relational operators and combine the results of multiple comparisons via Boolean logic or conditional statements.

Additionally, students were required to override the `getBookInfo` method by calling the superclass method, and to demonstrate the ability to construct a `String` containing multiple pieces of information.

Sample: 2A
Score: 8

Point 1 was earned for the correct class header, `public class Textbook extends Book`, which satisfies the requirement that `Textbook` be a subclass of `Book`. Point 2 was earned because the response contains a proper constructor header with three parameters of type `String`, `double`, and `int`, in that order, conforming to the requirements of the sample code. Point 3 was earned because the first line of the constructor is a call to the superclass constructor, and the arguments to that call are the first and second parameters of the constructor, `ti` and `pr`. The design uses inheritance by utilizing the constructor of the superclass to deal with book title and price information. Point 4 was earned because there is a single `private int` instance variable (`edition`), used to store the edition number and initialized with the value of an `int` parameter of the constructor (`ed`). Point 5 was earned because the response properly declares headers for the listed methods `canSubstituteFor`, `getEdition`, and `getBookInfo`, each with `public` access modifier and appropriate return type and parameter list, conforming to the sample code and corresponding results in the prompt. Point 6

Question 2 (continued)

was earned because the `getEdition` method returns the value of the instance variable `edition`. Point 7 was not earned because the response only checks for equal edition numbers, not edition numbers that are greater than or equal to those of the other book, as required in the specification. The response does correctly compare the titles of the two `Textbook` objects using `equals`, and it does use `&&` to combine the title comparison and the edition number comparison, but the incorrect comparison of editions caused this point to not be earned. Point 8 was earned because the overridden `getBookInfo` method properly calls `super.getBookInfo`, utilizing the inheritance design to get an appropriate `String`. Note that there is no accessor method for price information in the `Book` class, so it is necessary to call the `getBookInfo` method in order to obtain the `String` that contains title and price information, separated by a hyphen. Point 9 was earned because the response concatenates the results of the `super.getBookInfo` method call with the edition number and inserts a hyphen (`"-"`) between them.

Sample: 2B
Score: 6

Point 1 was earned for the correct class header declaration, including the inheritance from class `Book`. Point 2 was earned because the constructor header is properly declared, with three parameters of type `String`, `double`, and `int`, in that order. Point 3 was earned because the first line of the constructor is a call to the constructor of the superclass (`super`), and the arguments to that call are the first and second parameters of the constructor (`ti` and `pr`). Even though additional variables named `t` and `p` are declared and used for title and price information in other methods, point 3 only assesses the call to the superclass constructor. Point 4 was not earned because there are instance variables created to store the title and price, and those variables are used elsewhere in the class. This violates the class inheritance aspect of the design. Although an instance variable is declared and initialized with the parameter value for the edition, this point is about the declaration and initialization of only appropriate instance variables that are needed in this subclass. Point 5 was earned because the response properly declares headers for the listed methods `canSubstituteFor`, `getEdition`, and `getBookInfo`, each with `public` access modifier and appropriate return type and parameter list. Point 6 was earned because the `getEdition` method returns the value of the instance variable `e`. Point 7 was not earned because the response calls an `equals` method on the `int` value returned by `getEdition` to compare the editions. This is an error for two reasons: it is not valid to call `equals` (or any other method) on an `int`, and the comparison required in the problem is for greater than or equal, not exact equality. The response uses the instance variable `t` of the `Textbook` class, which it accesses as `this.t` (equivalent in this situation to `t` by itself), instead of calling `getTitle`. Because there is an accessible instance variable `t`, this is a valid access of a `String` that might hold a title. Even though that variable does not hold the correct information (and violates the inheritance design), that error was assessed elsewhere, and point 7 could have been earned if the method were otherwise correct. Point 8 was not earned because the overridden `getBookInfo` method does not call `super.getBookInfo` to get the title and price portion of the desired `String`. Point 9 was earned because the response concatenates a `String`, a `double`, and an `int`, with hyphens separating them. Although the variables `t` and `p` do not have values associated with them, `t` is a `String` and `p` is a `double`, indicating which is intended to contain title information and which is intended to contain price information. This point is about creating a `String` in a specified format out of multiple pieces of information, independent of the method and class designs.

Question 2 (continued)

Sample: 2C
Score: 2

Point 1 was earned for the correct class header declaration, including the inheritance from class `Book`. Point 2 was not earned because there is no constructor and no constructor header. Point 3 was not earned because there is no call to a superclass constructor. Point 4 was not earned because there is no initialization of an instance variable to contain the edition number, which should be initialized with information from the (missing) constructor. Part of any class design involves specifying how objects of the class are to be constructed, and an entirely omitted constructor will, in general, lose all points related to the constructor and initialization of instance variables. Point 5 was not earned because the parameter type in the `canSubstituteFor` header is a `String`. The method is intended to compare this object with another `Textbook` object, and not just a title `String`, so the method's parameter must be a `Textbook`. Even though the `getBookInfo` method has a proper header, all declared headers must be correct in order to earn this point. Point 6 was not earned because there is no method that returns the edition. Point 7 was not earned for multiple reasons. First, the comparisons are incorrect, in both the extraction of substrings (which are not useful for this algorithm and may also cause out-of-bounds errors due to the use of `title.length() - 4`) and in the use of `==` to perform the `String` comparison. Second, the response directly accesses the `private` instance variable `title` from the superclass without calling the `getTitle` method. If `title` had been declared as an instance variable in the `Textbook` class, accessing `title` directly would not be considered a problem for this point, and point 7 could have been earned if the method were otherwise correct. Point 8 was not earned because the overridden `getBookInfo` method does not call `super.getBookInfo` as required in an inheritance-based design. Point 9 was earned because the response concatenates a `String`, a `double`, and an `int`, with separators between them. The variables `title` and `price` are `private` in the superclass, so this method would not be able to access them, but improper access is not assessed in this point. This point is about creating a `String` in a specified format out of multiple pieces of information, independent of the method and class designs.