AP CS A Exam
Exam Overview & Hints!

Section I: Multiple Choice
40 Questions | 1 Hour 30 Minutes | 50% of Exam Score

- The multiple-choice section includes mostly individual questions, occasionally with 1–2 sets of questions (2 questions per set).
- Computational Thinking Practices 1, 2, 4, and 5 are all assessed in the multiple-choice section.

Section II: Free Response
4 Questions | 1 Hour 30 Minutes | 50% of Exam Score

- All free-response questions assess Computational Thinking Practice 3: Code Implementation, with the following focus
  - **Question 1:** Methods and Control Structures—Students will be asked to write program code to create objects of a class and call methods, and satisfy method specifications using expressions, conditional statements, and iterative statements.
  - **Question 2:** Classes—Students will be asked to write program code to define a new type by creating a class and satisfy method specifications using expressions, conditional statements, and iterative statements.

- **Question 3:** Array/ArrayList—Students will be asked to write program code to satisfy method specifications using expressions, conditional statements, and iterative statements and create, traverse, and manipulate elements in 1D array or ArrayList objects.

- **Question 4:** 2D Array—Students will be asked to write program code to satisfy method specifications using expressions, conditional statements, and iterative statements and create, traverse, and manipulate elements in 2D array objects.

Even though not explicitly listed, Strings are very popular on the Free Response part of the AP exams. Know the string methods listed on the reference sheet really well!

Java Reference Sheet

or here:

https://apcentral.collegeboard.org/pdf/ap-computer-science-a-java-quick-reference.pdf

Strings

Strings and String methods have shown up every year. Likely it will show up on this year's AP Exam.

You may have to work with 1D array or ArrayList of Strings.

You should know how to use **substring(int beg, int end), substring(int beg), indexOf, equals, length and compareTo** methods. See the College Board reference sheet.

Free Response

General Strategies for FRQS:

1) Read the questions before you read the code! This will help you focus.

2) Unless a question specifically addresses efficiency, focus on clear, simple code rather than complicated, efficient code.

3) Comments are not necessary. Use only to organize your thoughts.

4) Pay attention to the code they give you:

- Do you have more than one class? Use it!
- Did they give you methods? You will need them!
- Are there instance member variables that you should use? If so, use them. Do not re-declare.
- Only use accessor (getter) and mutator (setter) methods to access private instance member variables.
- Do not re-write or change method headers.
- Use parameters (names and types) as given

Free Response

General Strategies for FRQS:

5) If part (b) references a method written in part (a), you will likely need to use this method. If it is not clear to you how, stop and think before you write.

6) Pay attention to the return type. If it is not void, be sure to include a return statement!

7) If a method comment says /* Implementation not shown */. You do not to implement the method. But you'll likely need to use the method in your code at least once.

Free Response

General Strategies for FRQS:

8) Write down some code for every question! Leave Nothing Blank! Get partial credit if not full credit!

- does it need a loop?
- an if statement?
- return statement or assignment statement?

Example 1:

```
public double funMethod(){
     double result;
     …
     return result;
}
```

More Examples

Leave Nothing Blank! Get partial credit if not full credit!

```
public int[] funMethod(int length){
     int[] list = new int[length];
     …
     return list;
}
public ArrayList<Bug> funMethod(){
     ArrayList<Bug> buggies = new ArrayList<Bug>();
     …
     return buggies;
}
```

More Examples

Leave Nothing Blank! Get partial credit if not full credit!

```
public boolean funMethod(int something){
     …
     if(something > somethingElse)
          return true;
     // must have a default return; every return can't be  //
locked in an if
     return false;
}
```

Question 1: Methods and Control Structures
See previous years' similar FRQs:
2018 #1, 2019 #1, 2017 #3, 2018 #2, 2021#1

Methods and Control Structures

Question 1 of the FRQ will likely ask you to write methods that utilize control structures(for vs while loops, conditionals).

You are given (usually static) methods with parameters and will be asked to write a method that calls one of the given methods in its implementation.

If a method comment says /* Implementation not shown */. You do not to implement the method. But you'll likely need to use the method in your code at least once.

AVOID using arrays/arraylists for this problem.

2019 #1 is a good example of this problem.

Methods and Control Structures

Likely you will need to use for loops and if-else if-else conditionals.

Likely your method will return some value. See its return type in its header.

Using Strings and its methods can be required here in this question(or Question 2). See for example 2017 #3 for a String question that fits "Methods and Control Structures" question.

You should know how to use **substring(int beg, int end), substring(int beg), indexOf, equals, length and compareTo** methods. See the College Board reference sheet.

`String` methods

Method name	Description
`String(String str)`	Constructs a new String object that represents the same sequence of characters as str
`int length()`	Returns number of characters in this string

`substring(`**`index1, index2`**`)`
or
`substring(`**`index1`**`)`
	Returns the characters in this string from index1 (inclusive) to index2 (exclusive);
	if index2 is omitted, grabs till end of string

`boolean equals(String other)`	Returns true if this is equal to other; returns false otherwise
`int compareTo(String other)`	Returns a value < 0 if this is less than other; returns zero if this is equal to other; returns a value > 0 if this is greater than other
`indexOf(`**`str`**`)`	Returns index where the start of the given string appears in this string (-1 if not found)

`String` method examples

```
// index      01234567890123456789
String s1 = ”programming in java";

System.out.println(s1.length());
// 19
System.out.println(s1.indexOf(“i"));  // 8
System.out.println(s1.indexOf(“gram"));  // 3
System.out.println(s1.indexOf(“hi"));  // -1

System.out.println(s1.substring(7, 10));  // ”min"
System.out.println(s1.substring(12));  // ”in java”
System.out.println(s1.substring(2,3));// ”o”

String s2 = s1.substring(10, 17);  // ”g in ja”
```

`indexOf` Example

indexOf can be confusing. Here's an example.

```
public static mystery(String str){
        String answer = "";
        for(int i = 0; i < str.length(); i++){
             String currentLetter = str.substring(i,
  i+1);
             if("aeiou".indexOf(currentLetter) != -1)
                  answer += current;

    }
        return answer;
  }
```

What is the output?

```
System.out.println(mystery("cat in hat"));
```

Answer: aia

Do Problem 2018 #2

Complete 2018 #2.

This is a good problem that uses 1D array, Arraylist, and Strings and is a good Question 1(Methods/Control Structures) type of problem.

Also do the following 2018 #1, 2019 #1, 2017 #3, 2021 #1 to prepare for Question 1.

Question 2: Writing Classes

See previous years' similar FRQs:

2019 #2, 2021 #2, 2016 #1, 2015 #2, 2021#2

Question 2

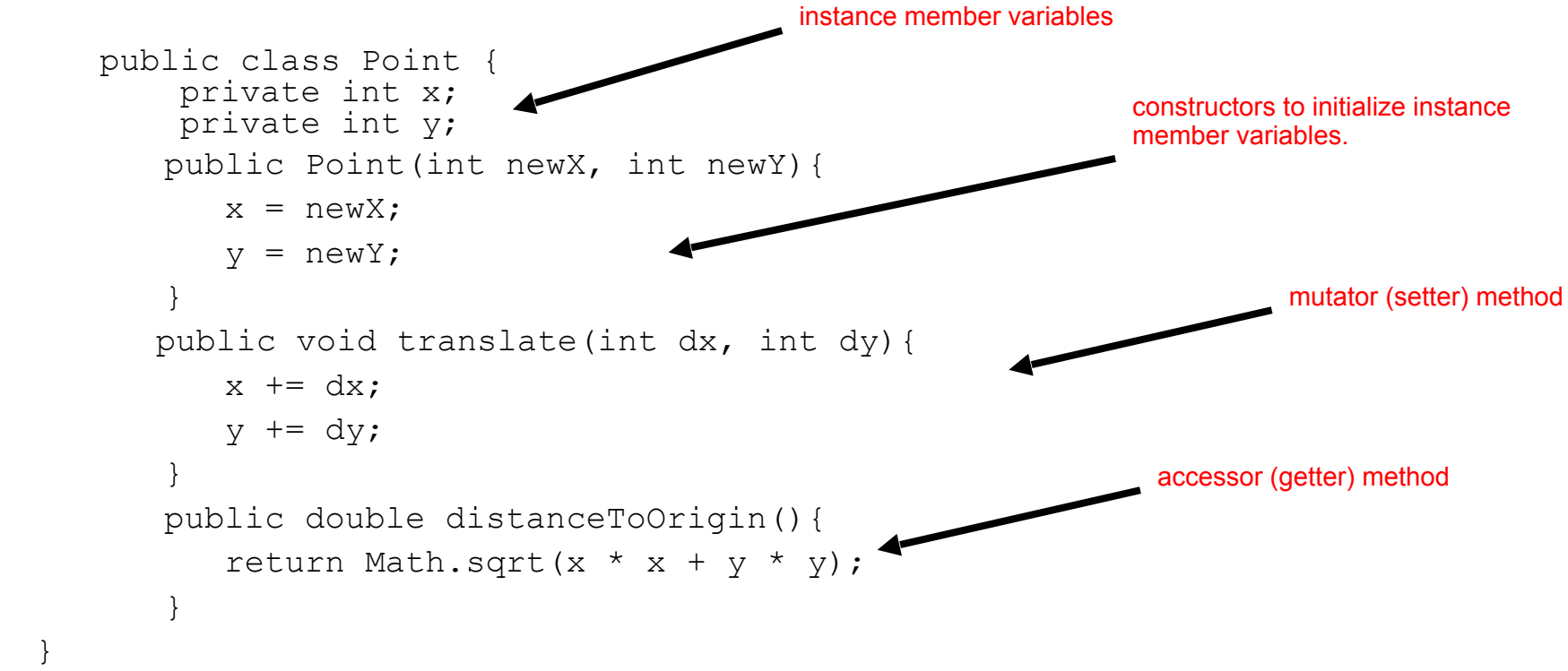
This question will ask you to write an entire class including private instance member variables, constructors and methods.

The class implementation typically involves 2 – 4 instance member variables, one constructor and 2 – 3 instance member methods.

AVOID using arrays/arraylists for this problem. Otherwise you'll be doing needless for loops. Consider a variable(int or double) that accumulates(sums) values rather than keeping track of each value in an array/arraylist.

The Point class' implementation on the next slide provides an example of the anatomy of a class.

An Implementation of Point




```
public class Point {
    private int x;
    private int y;
   public Point(int newX, int newY){
      x = newX;
      y = newY;
   }
   public void translate(int dx, int dy){
      x += dx;
      y += dy;
   }
   public double distanceToOrigin(){
      return Math.sqrt(x * x + y * y);
   }
}
```

Typically, the question will give you examples of method calls and corresponding outputs and require you to create a class that satisfies those specifications. 2019 # 2 is a great example of this.

explains you how to write your constructor!

Statements and Expressions	Value Returned (blank if no value)	Comment
`StepTracker tr = new StepTracker(10000);`		Days with at least 10,000 steps are considered active. Assume that the parameter is positive.
`tr.activeDays();`	`0`	No data have been recorded yet.
`tr.averageSteps();`	`0.0`	When no step data have been recorded, the `averageSteps` method returns `0.0`.
`tr.addDailySteps(9000);`		This is too few steps for the day to be considered active.
`tr.addDailySteps(5000);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`0`	No day had at least 10,000 steps.
`tr.averageSteps();`	`7000.0`	The average number of steps per day is (14000 / 2).

And what methods to implement!

StepTracker

Students have implemented correctly StepTracker(2019 #2) using an ArrayList. **However, avoid using ArrayList/1D Arrays or 2D arrays for #2.**

Otherwise, you'll be doing needless for loops in your implementation.

The previous problem(2019 #2) is simple problem that involves only working with one class.

Another problem that is slightly more complex is 2021 #2. This problem involves writing a class that uses ANOTHER given class.

In this problem, you are given a SingleTable class with accessor (getter)s and mutator (setter)s methods and you are asked to write the CombinedTable class that uses the SingleTable class.

Try this problem. Here's a hint: A subtle error to this problem is not using SingleTable instance member variables! If you use int and double instance member variables, your answer will not receive full credit because of:

`t2.setViewQuality(80);`		Changing the view quality of one of the tables that makes up `c2` changes the desirability of `c2`, as illustrated in the next line of the chart. Since `setViewQuality` is a `SingleTable` method, you do not need to write it.
`c2.getDesirability();`	`67.5`	Because the view quality of `t2` changed, the desirability of `c2` has also changed.

Question 2

Thus, the take away is that if the problem gives you a class(SingleTable) and requires you to use it another class(CombinedTable), it is likely that you will need to make at least one instance member variable(for example, SingleTable t1) in the class that you are writing(CombinedTable).

Please do the following 2019 #2, 2021 #2, 2020 #2, 2016 #1, 2015 #2 and check your solutions to prepare for Question 2.

Question 3: Arraylist/1D Array

See previous years' similar FRQs:

2010 #1, 2013 #1, 2017 #1, 2018 #2, 2021#3

ArrayList/1D Arrays

We will first prepare for the FRQs by reviewing ArrayList and 1D arrays. This will be Question 3 on the Free Response.

From previous years' exams, it is likely that Question 3 is an ArrayList question instead of/in addition to an 1D array question.(ArrayList or both, unlikely to be just a 1D array question)

See the following problems 2010 #1, 2013 #1, 2017 #1, 2018 #2

It's very common to have a class(e.g Student) and another class which contains either a 1D array of Student objects or an ArrayList of Student objects.

```
public class Student {
        …
        public String getName(){..}
        public double getGpa(){…}
}
```

ArrayList/1D Arrays

```
public class Course {
        private ArrayList<Student> students;

    public Course() {
            students = new ArrayList<Student>();
         ...
    }
}
```

OR

```
public class Course {
        private Student[] students;

    }
}
```

Traversing 1D Array vs. ArrayList

Arrays:

```
double sum = 0;
for(int i = 0; i < students.length; i++){
     sum += students[i].getGpa();
}
```

ArrayLists:

```
double sum = 0;
for(int i = 0; i < students.size(); i++){
     sum += students.get(i).getGpa();
}
```

Traversing 1D Array vs. ArrayList

Use for each loops if you are only traversing the arraylist and not removing items from it.

Both Arrays and Arraylists:

```
double sum =0;
for(Student s: students){
     sum += s.getGpa();
     // notice no [i] and no get(i)!!
}
```

ArrayList

Remember for Arraylists that if you remove an item, items to the right of it will shift left!

And if you add an item to the list, items will shift right to accommodate!

Make sure to take this into account when you are adding or removing items.

ArrayList/1D Array Tips

When using the ArrayList method remove(), remember you can only REMOVE BY INDEX (not by object), so you must use a traditional for loop (and NOT a for each loop).

```
for(int i=0; i<list.size(); i++){
        if(list.get(i).getScore() <= 50)
                list.remove(i);
                i--;//AND don't forget to decrement the index!
}
//OR traverse the list backwards, i.e.
for(int i=list.size()-1, i>=0, i--){
        if(list.get(i).getScore() <= 50)
                list.remove(i--);
}
```

ArrayList/1D Array Tips

Avoid array and ArrayList out-of-bounds exceptions! Especially when comparing consecutive elements.

Example: You need to compare consecutive elements to determine if the values are increasing.

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

Finding the smallest or largest

When you need to return an element in a list (i.e. largest, smallest,…), assign the initial element to the **first one** in the list.

Example : You need to find the Dog with the most fleas.

```
//returns the Dog object in pack with the most fleas
public Dog mostFleas(ArrayList<Dog> pack){
     Dog most = pack.get(0);
     for(Dog dog : pack)
          if(dog.getNumFleas() > most.getNumFleas())
               most = dog;
     return most;
}
```

Manipulating an Array using Arraylist

An ArrayList can be helpful when you are trying to manipulate an array (i.e. remove items, rearrange in random order, etc.).

Example:

If you need to remove several objects from an array based one some condition, a strategy would be to save all the objects to an ArrayList and remove the objects before storing the remaining objects back to the array.

See code on the next slide.

Manipulating an Array using Arraylist

```
//returns a new array with bad things removed
public Thing[] removeBadThing(Thing[] widgets){
     ArrayList<Thing> good = new ArrayList<Thing>();
     for(Thing wid : widgets){
          if(!wid.isBad()) // i.e good
               good.add(wid);
     }
     Thing[] goodOnes = new Thing[good.size()];
     for(int i=0; i < goodOnes.length; i++)
          goodOnes[i] = good.get(i);
     return goodOnes;
}
```

Question 3

Please do the following 2010 #1, 2013 #1, 2017 #1, 2018 #2, 2021 #3 and check your solutions to prepare for Question 3.

Question 4: 2D Array
See previous years' similar FRQs:
2019 #4, 2018 #4, 2017 #4, 2016 #3, 2021#4

Row Major Order Traversal

Example : Count the number of Students objects whose GPA is above a threshold. (row major order)

```
public class Course{
     Student[][] students;
     // constructors not shown
     public int aboveThreshold(double threshold){
          int count = 0;
          for(int row = 0; row < students.length; row++){
            for(int      col = 0; col < students[0].length; col+
+){
               if(students[row][col].getGpa() > threshold)
                    count++;
            }}
          return count;
     }
```

Column Major Order Traversal

Example : Count the number of Students objects whose GPA is above a threshold. (column major order)

```
public class Course{
     Student[][] students;
     // constructors not shown
     public int aboveThreshold(double threshold){
          int count = 0;
          for(int col = 0; col < students[0].length; col++){
            for(int      row = 0; row < students.length; row++){
               if(students[row][col].getGpa() > threshold)
                    count++;
          }
          return count;
     }
```

Along One Row

Example : Count the number of Students objects whose GPA is above a threshold in the given row.

```
public class Course{
     Student[][] students;
     // constructors not shown

     public int aboveThreshold(int row, double threshold){
          int count = 0;
          for(int col = 0; col < students[row].length; col++){
               if(students[row][col].getGpa() > threshold)
                    count++;
          }
          return count;
     }
```

Along One Column

Example : Count the number of Students objects whose GPA is above a threshold in the given column.

```
public class Course{
     Student[][] students;
     // constructors not shown
     public int aboveThreshold(int col, double threshold){
          int count = 0;
          for(int row = 0; row < students.length; row++){
               if(students[row][col].getGpa() > threshold)
                    count++;
          }
          return count;
     }
```

Major Diagonal Traversal

Example : Count the number of Students objects whose GPA is above threshold along the major diagonal. Assume 2D array is square.

```
public class Course{
     Student[][] students;
     // constructors not shown
     public int aboveThreshold(double threshold){
          int count = 0;
          for(int i = 0; i < students.length; i++){
               if(students[i][i].getGpa() > threshold)
                    count++;
          }
          return count;
     }
```

Minor Diagonal Traversal

Example : Count the number of Students objects whose GPA is above threshold along the minor diagonal. Assume 2D array is square.

```
public class Course{
     Student[][] students;
     // constructors not shown
     public int aboveThreshold(double threshold){
          int count = 0;
          int dim = students.length;
          for(int i = 0; i < dim; i++){
               if(students[i][dim - 1 - i].getGpa() > threshold)
                    count++;
          }
          return count;
     }
```

Question 4

Please do the following 2019 #4, 2018 #4, 2017 #4, 2016 #3, 2021 #4 and check your solutions to prepare for Question 4.