NOTE: THIS FAQ WILL BE UNSHARED DURING THE PERIOD OF MAY 15-22. PLEASE MAKE A COPY IF YOU WOULD LIKE TO USE IT AS A RESOURCE

[WIP] Guide to the AP CSA Exam

Free to share, but teachers, if you use this guide for in your class, credit is needed :)
Questions? Comments? Corrections? Email: melodyc.lam@cms.k12.nc.us

Table of Contents

Updates

8/21/2019:

- crossed out obsolete material (left in for historical reasons)
- **2020 Exam Updates added (CTRL+F "2020 Exam Update" to find these.)**

5/7/2018:

- **last minute tips for the exam, including what to bring.**
- Added Autoboxing, wrapper classes, and a warning about using list[i] for ArrayLists (This is a Java exam, not a Python exam! Also, it's a penalty.)

4/30/2018: Added == vs .equals() & String pooling discussion. Because I cannot believe how many people do x == y for Strings (please don't.)

4/19/2018:

- Edits for clarity:
    - String is now capitalized when I'm referring to the type
    - list -> array/ArrayList, unless specific to a structure
- Added differences between array/ArrayList to vocab document

- Added visual table for differences between array/ArrayList to vocab document
- **TO DO: moar pictures**
- Added Acorn Book & FRQ links to General Information

4/18/2018: Edited for clarity. Also, I'm alive.
1/24/2018: Added new things to the vocab list ("constructor", "call", "static")

General information about this document

This document is designed to help CSA students with navigating the CSA exam and its style. I thought that this would be a good way to give everybody a level playing field coming into the exam.

An AP exam should never be a "surprise" to anybody. The AP exam is designed to test one's knowledge of the concepts that are presented in the AP curriculum. There are never any "trick" questions that cover material that isn't in the curriculum documents.

This document should not AND IS NOT a replacement for actual study and practice. This is a guide to how the exam works, how the MC works, what is covered on the FRQs, etc. This is not a "cheat sheet" you can print out and take to the exam room to study before the exam. (**On the day of the exam, you should not study. If you do not know it by that morning, you won't know it for the exam. Don't try.)**

General documents you should at least take a glance at:

~~"Acorn Book" -- especially pages:~~

- ~~Sample questions: starting on page 21/83~~
- ~~AP Java subset (basically, what do you need to know in Java): page 65/83~~
- ~~Quick Reference Sheet (the one you get on the exam): page 71/83~~

2020 Exam Update: The CED (aka the NEW acorn book):

- Sample Questions: page 191/198
- Quick Reference Sheet: page 209/216
- (AP Java Subset) Exam Weighting: page 188/195

FRQs -- don't use the AP Student website, which makes you log in.

General information about the AP CSA Exam

The AP CSA Exam is in two parts: the multiple choice, which has 40 questions (5 choices), and the FRQ, which has 4 questions, all of which involve multiple parts (or writing a whole class).

Each portion is timed at 90 minutes, with a 10 minute break between the two parts. The multiple choice works like all the other AP Exams -- you bubble in your answers on the answer sheet. Make sure that you are leaving ample time to put answers down if you are someone who circles answers in the booklet and then bubbles them in at the end. Both booklets are yours to write in (MC & FRQ); the FRQ booklet is sent for grading.

The Quick Reference: The Quick Reference is printed in the front/back of the booklet and **cannot** be torn out. Tearing out pages from the booklet is a misadministration and can cost you your score, so **don't do this. You do get the Quick Reference for both portions of the exam.** It is to be used as a tool, not a crutch --

familiarize yourself with the QR, but do not memorize it. If you don't know what those functions do now, you won't know it for the exam.

No, you don't get a calculator. However, the mathematics should be quite simple (grade school arithmetic + modulus + absolute value sometimes) and you should not need one.

~~**In past years, there have been base conversion questions.** However, there's only 1-2 and if you're getting stuck on those, just skip them.~~
2020 Exam Update: As of the 2020 exam, there are no more base conversion questions. Rejoice!

The FRQ: ~~Unlike most exams, this exam has only 1 FRQ booklet where you write all your answers.~~ **New for 2020: 2 booklets. One with the question descriptions, one booklet of blank paper where you write the FRQ. This is very similar to other exams where you bubble which question you are answering on the page then write the answer on that page.** You *should* have ample space to write the FRQ answer on the space provided, but you're allowed extra paper to write your answers in.

Please do not write your FRQ in pen. Use a pencil. Seriously. You will be making mistakes and rewriting code.

If your answer for an FRQ part does not fit on the page, you're probably doing it wrong.

I made a big mistake/didn't put a line of code when I should've: cross out the old answer and write your new answer as clearly as possible, drawing arrows if you have to. Draw arrows to the place where you are inserting code. **Readers will not read any code that is crossed out.** Don't waste your time erasing (unless it's a quick fix).

Supplies to bring (good for any AP exam, really):

- **ID. ID. ID. Photo ID issued by school or government.** Do not be that person that can't take the exam because you forgot one! **Passports work!**
- **Multiple WOODEN pencils, sharpened.** Ticonderoga pencils are amazing and rarely break. Make sure that you have them sharpened for the exam, ready to go.
- **Pencil sharpener.** Go to Staples and get a sharp pencil sharpener.
- **Blue or Black ink pen**. You need one to complete the front of the FRQ booklet.
- **Erasers:** get a Statleder eraser or a Japanese eraser (like Uni Boxy or Tombow Mono). The pink ones may rip the paper and you don't want that to happen.
- **Jacket**. Useful if you know ahead of time if it's going to be freezing in your room. And it's easy to take off when you start sweating.
- **Watch, analog or digital**. Do not bring Apple Watches, Fitbits, Android watches, or anything that connects to the Internet. Do not bring anything that makes an annoying BEEP sound that can't be turned off.
- **FOOD.** You can keep it outside the exam room and use the 10 minute break between portions to snack. **EAT LUNCH (AND ENERGY DRINKS DON'T COUNT!!!)**

Vocabulary used in this document & on the AP exam

Class: A class is a blueprint for variables and behaviors, called **methods**. If you imagine Java as a factory, you feed Java the class and it will produce **objects** that can be manipulated individually. This is the crux of **object-oriented programming.**

Objects: Objects are **instances** of a class, and they are **instantiated** using the class's **constructor**. Because objects are separate from each other, an object can have a state that is different from another object in the class. Examples of objects are the Object class (the granddaddy of all objects), String, and ArrayList.

Variables: also called **primitives,** these are as follows: int, double, boolean, char. String is an **object** because there are methods associated with Strings. Variables do not have methods associated with them.

Instantiation: the process by which you create an object, using the constructor and the **new** keyword.
Example: `Object o = new Object();`

Declaration: making the actual class in question, or when you declare that you're making something.

Initialization: giving a variable a value (like int x = 5 or String n = new String("Hello");)

Call: when you want an object to perform the task in a method; the call comes from the object itself using dot notation.
Examples: If I have a string named x and I want a substring from 0 to 5, then the call would be `x.substring(0,5)`

Instance variables: Variables that an object has. These are variables that are normally **private** and can only be changed through methods (accessors and mutators, or as I like to call them: **set** and **get** methods.)

Constructors: methods (some sources say they aren't methods, but I consider it one because it does something) that **instantiate an object**. The constructor has **no return type** (it's not void) and it is always the **same** name as the class!
Constructors are almost always overloaded! The default constructor always exists, even if the default constructor is not written and one can always call a constructor for a class (unless the constructor is private, like in the Math class)

Accessors: also called **get** methods, because they usually have the prefix get_____(). These methods return the state of the given instance variable, and normally have the return value of whatever you're trying to get.

Mutators: also called **set** methods, because they usually have the prefix set_____(). These methods are usually void and allow you to set the instance variable to the parameter inside the parentheses.

Iteration: the process of looping through something.

Conditional: the Boolean statement after an if-statement.

Reference: The actual location where an object's data is stored. The **value** of a Java object is actually its reference (!!!!) and not the data itself.
Example: let's just say I make a object called `mazdaRX8` that is of the Car class. Then I call `System.out.println(mazdaRX8)`. What prints out is a hexadecimal string, which represents the object's location in memory, and not the actual data/instance variables included in the Car class.
This is why you need to **override** the `toString()` method in every class that you make. This method is inherited from the Object class.

Overload: Methods that have the **same name** in the **same class** that have **different parameters**. The parameter order/number must be distinct from the other methods with the same name.

Override: Methods that have the **same name** in **different (inherited) classes** that have **the same parameters.** When you override a method in the child class, you're changing the method's behavior for that class. See the example about toString() above.

Scope: there are two kinds of scope: local and global/instance. The scope of a variable is defined by what curly braces the variable lives in.
Example: In the following class, x is a local variable and age is an instance variable. I've highlighted the braces to show you which scope is which.

```
public class Example {
      int age;

public void setAge(int x) { age = x; }
}
```

Static methods: static methods are also called functions in other languages. Static methods are always called from the class, not an instantiated object. Examples include the Math class & the Arrays class.
Ex: `Math.abs(200), Math.sin(x)`

Array: An array is a fixed-size list of **LIKE/SAME** items. Arrays **can** hold primitives. Think of them as cubby holes in a kindergarten class -- only one item can go in each **element** of an array.
Here's an example of **instantiating an array:**
`int[] array = new int[10];`

Notice that instantiating an array uses the word **new**. This is because an array is **an object**, but does not have any methods associated with it!! Weird, huh? Because we instantiated the array with the word "int" before the name, this means that this array can only hold integers. Trying to put a String into this array would cause a syntax error.

To access elements of an array, use the array notation like so: `array[x]`, where x is between 0 and array.length-1. Notice that array.length is **NOT** a method. (It's a public field.)

To change elements of an array, use the array notation and make the element equal to a value. For example: `array[1] = 5` would change the element at position 1 to the number 5.

- **You can use the enhanced for loop on an array.**

ArrayList: An ArrayList is a dynamic list of **LIKE/SAME** items. ArrayLists **cannot** hold primitives -- **ONLY OBJECTS!**

To instantiate an ArrayList:
`ArrayList<Type> list = new ArrayList<Type>();`

(Just FYI not specifying <Type> defaults the objects in the ArrayList to belong to the class Object...which is not a good thing.)

~~Often you'll see this instead:~~
~~List<Type> list = new ArrayList<Type>();~~
~~**That's because List is actually an interface. It also forces you to use only the methods on the Quick Reference.**~~

To retrieve items from an ArrayList:
`list.get(index)` ***** **USING THE ARRAY NOTATION ON AN ARRAYLIST IS A PENALTY** *****

To **add** items to an ArrayList (add to the end/append to the end):
`list.add(item)`

To change an item in an ArrayList:
`list.set(index, what you want to set it to)`

To remove an item in an ArrayList:
list.remove(index)

Be careful when you remove items from an ArrayList. Indices will shift to the left because of removal (Java will automatically "fill in" the holes created because of removal.) If you are removing items from an ArrayList, traverse the ArrayList backwards -- because you're going from right to left, your algorithm will handle the shift gracefully. Otherwise, you would have to remember to shift the counter that you're using to traverse the list to the left by one every time you remove.

DO NOT USE THE ENHANCED FOR LOOP TO REMOVE ITEMS FROM AN ARRAYLIST. (This is because the enhanced for loop is really using an Iterator object and you cannot remove items from a structure that is using an iterator to traverse the structure.)

Here's a visual for Array/ArrayList differences.

== VS .equals() -- what's the difference?

- **== is used for comparing reference values of the objects. You always use == for primitives.**
- **.equals() is used for comparing the content of the objects. (Objects must have an equals() method for you to do this.)**

Example:
String x = "Hello";
String y = new String ("Hello");

System.out.println(x == y) // prints false
System.out.println(x.equals(y)) // prints true

String pooling
Java has the ability to combine Strings (and only Strings!) together that are the same into one reference. Here's an example.
String x = "Hello";
String y = "Hello";

x and y both refer to the same object! So if I asked x==y, this would return true (not false)!

To force Java to make x and y distinct objects, you need to use the String constructor:
String x = "Hello";
String y = new String ("Hello");

AutoBoxing/Wrapper classes: On a technical level, whenever you operate on an ArrayList full of Integers or Doubles (or any structure that requires object references), Java will automatically convert the primitives `int` & `double` into `Integer` & `Double` for you. In earlier versions of Java you needed to explicitly do this using the Integer and Double constructors. Integer & Double are called **wrapper classes**. The ability of Java to do this wrapping implicitly is called **autoboxing**.
2020 Exam update: You need to know how to use the Integer and Double constructors. You also need to know how to use the methods `intValue()` and `doubleValue()` to retrieve the primitive version of the corresponding objects.
You do not need to know what values Integer.MAX_VALUE & Integer.MIN_VALUE actually hold (it's 2^31-1 and -2^31 respectively).

The Multiple Choice

You will have 2 minutes and 15 seconds per question. That does not mean that you can spend 2 minutes and 15 seconds per question.

In fact, if you are spending that amount of time on questions 1-10 on a practice exam you should seriously re-think your strategy. The thing is, if you can't complete questions 1-10 in around 10 minutes, you're going to run out of time for the last 10 questions or so on the exam. You'll want to aim for getting at least 35/40 questions right on the exam, if possible. That seems like a high bar, but it's achievable.

The questions will test you over your knowledge of Java code and concepts. Most questions require reading and parsing code snippets, or matching a code snippet to a problem. A lot of the code should be familiar to you!

Algorithms you should know by heart and be able to recognize on the exam (roughly from most important to least important):

- counting algorithm (count items from an array/ArrayList)
- linear search algorithm
- swapping algorithm (swap two variables)
- parsing a String using a for loop
- parsing an array/ArrayList using a for loop/enhanced (for each) loop

- removing items from an ArrayList (see 2010 #1: MasterOrder)
- binary search algorithm
- selection sort algorithm
- insertion sort algorithm
- factorial algorithm (recursive and non recursive)

Note on selection sort and insertion sort: You will **not** ever write these algorithms on the FRQ, but at least 2-3 questions will be on these two algorithms. They will be given to you on the exam in the question text (probably) and you will be told what the algorithm does.

Recursion: At least 3 of the questions will be on recursion. These questions will normally be somewhere starting at question 25, but they might come earlier. Make sure that you watch your time! If you're spending too much time on the recursion questions, move on. Answer questions you know.

Override vs. Overload: There will be a question about this and most students will confuse the two. Do not confuse the two (see above for the difference)

The optimal strategy:

- Make a pass through the exam. Answer questions that you know how to do. Skip (and star) questions you can't do without more effort. Skip the recursion questions on the first pass unless you can quickly figure out the answer or know the algorithm.
- After the first pass, check the time and make a second pass through the exam. This way, you can spend more time on the more difficult questions, like nested loops, matrices, etc. You can eliminate answer choices very easily -- often times you can whittle it down to 2 and make a guess.
- **Guesses are not penalized. It is in your best interest to answer all the questions.**

The FRQ

You will have 22 minutes and 30 seconds per FRQ question. This does not mean spend 22 minutes and 30 seconds per question.

~~Just like the MC, the FRQ is roughly in difficulty order, although there have been exceptions (see FRQ year-by-year analysis below).~~

~~Every year, the FRQ will test you over **at least** these things:~~

- ~~String manipulation~~
- ~~Arrays/ArrayLists~~
- ~~Matrices~~
- ~~Class creation/writing (you will need to write a whole class)~~
- ~~Inheritance/Interfaces/Abstract Classes~~
- ~~Searching a String, array, and/or ArrayLists **(linear search ONLY)**~~

2020 Exam update: The FRQ types and questions are now explicitly defined.

- **FRQ #1: Method and Control Structures (can you create objects & call methods from a predefined class?)**
- **FRQ #2: Class (class writing)**

- **FRQ #3: Array/ArrayList (possibly both?)**
- **FRQ #4: 2D array**

Fundamentally, this is not a change from previous years! These things have always been tested in some way...it's just now spelled out for you.

All FRQs will be in at least 2 parts, sometimes 3 parts, unless it's the class writing question, in which case there is only one part: write the **whole** class. Hopefully you have practiced writing code as part of labs and on FRQs...if you haven't, you need to start **now.**

In general:

- **There is no reading period, unlike other exams with a considerable reading portion.** The 90 minutes includes time to read and write your responses.
- If the FRQ has methods, **either you're writing the method or you're using the method in the problem somewhere.** These will be highlighted in grey! At no point does the exam give you a method that you will never use/write in some way.
- **Oftentimes, part B/C will involve using the method that you wrote in the previous parts. Do note that re-writing code that was implemented in a previous part will cost you points.** If you do not know how to write the method in part A and you use the method in part B, this will **not** cost you points (in other words, the exam features error carried forward/no double jeopardy.)
- **If you make a mistake, ~~try your best to erase as best you can and re-write the code~~ just cross it out....if you've run out of time, making an arrow to where your code should be will help the reader understand your code.**
- **Make sure that your handwriting is as best as you can, and that CAPITALIZATION MATTERS.**
- **DO NOT USE PACKAGES/METHODS THAT ARE NOT ON THE QUICK REFERENCE.** This includes the Arrays and Random packages. No, you won't get penalized for using them **BUT** this increases your chance of having your code be scrutinized even closer, which can cost you points.
- **It is HIGHLY recommended that you avoid recursive answers on the FRQ.** Remember that any recursive answer can be done in a non-recursive way. The reason why I do not recommend it is 1) it may take longer to figure out how it's done recursively 2) with the time constraint, it's easy to forget to write something like the base case of a recursive function, which will cause a loss of points.
- ~~**Lisa needs braces**~~ **Please be as clear as possible on the FRQ.** While, yes, you don't always need braces when writing if-statements or loops, it's best to put the curly braces in. It's clearer to the reader what your intent is. Likewise, make sure that your variable names are appropriate. Sure, int x is cool, but maybe a better way of making a variable is to state what it represents. (In for loops, it's okay to use i and j as counter variables, though.) **And yes, this isn't Python, but please indent your code and write properly styled code (do not put it all in one line because while you might find that funny, the reader will not!)**
- **Minor mistakes won't matter.** Missing a semicolon once or forgetting a close brace at the end of the method body won't hurt your score. However, if you consistently make a mistake you will be penalized. It helps to read the penalty scoresheet at the beginning of the 2017 FRQ rubric. **Do note that penalties may change from year to year, and certain questions have separate penalties that are assessed for that question only.**

Class Writing: When the question requires writing a class, make sure that you read **both** the text portion and the code examples. Each question will include both. Your class must be able to work with the sample code that is given.

There is a guarantee of having to write these portions of a class (in this order):

- Instance variables
- Constructor
- Methods of the class
- (that's pretty much what a class has)

Oftentimes you need to infer what the instance variables must be. One way to check what variables you need is to look at the parameters of the constructor given. If you write extra instance variables (when instance variables were not specified) then you will not be penalized, but don't just write variables randomly that you won't ever use!

Sometimes you may have to write a child class or ~~a class that implements an interface~~. **You will not be given a class header for class-writing questions. You must write the COMPLETE class. If you implement an interface/extend an abstract method, you must remember to write the methods included in the abstract class that were labeled "abstract".**

Things to watch out for:

- Are your parameters in the right order?
- Are your instance variables **private**?
- Have you written every required method that needs to be in the class (especially important for abstract methods!)?

Method Writing: the majority of questions on the FRQs will involve implementing and writing methods that are given to you. **The method header will be given to you when you write these problems. Do not write the method header on your paper.** Make sure that if a method returns a value, that somewhere along the way you write the return statement. A good way to not forget is to make a variable that represents the value to be returned, and then write "return <variable>" at the bottom of the page.

But I don't know how to write this FRQ, or in other words, how to get at least 1/9 out of your FRQ….

Do not be the person that gets a 0/9 on an FRQ! Every point counts and could mean the difference between getting credit in college and not (or for those of you who want AP Scholar, the difference between getting that and not getting it)
Each FRQ has "free points" where if you write something and it's valid, you'll get the point.
Here are some good ways to get "free points":

- Writing headers for methods when making a class.
- Writing the class header when making a class
- Making a variable that represents the value that you need and returning this value
- Writing a for loop that goes through a list, if the question involves searching a list.
    - Although, you should know how to search through a list!
- Writing a for loop that goes through a string