APCS Exam Study Tips

//Multiple Choice Tips

1. 40 multiple choice questions in one hour and 30 minutes – 2 min 15 seconds per question.
2. Read each question closely!  There is usually one word that defines (makes / breaks) the meaning of the question.
3. Use the white space area on each page to sort through any/all looping && recursion questions.
4. Don't forget to use the Quick Reference Guide which lists common methods – especially ArrayList and String methods.
5. Answer all questions – you are not penalized for guessing.
6. Wear a watch and monitor your time closely. You may not be given a 5 minute warning and it is important that you bubble in answers to each question.
7. Multiple choice practice exams:
* Georgia Tech AP Exam Website - http://ice.cc.gatech.edu/apexam/
* Amplify MOOC – APCS Review - https://course-mooc.amplify.com/
* Learnerator APCS - http://www.learnerator.com/ap-computer-science

**//Free Response Tips**

1. Four free response questions in one hour and 30 minutes – 22.50 minutes per question.
2. Don’t put any identifying information in the free response portion of the exam. That includes your full name, my name, or TJ’s name. Feel free to tell jokes/draw pictures. Keep them appropriate. Any words you write not related to the question should be in comments.
3. Read the intro to the overall question (just a few lines) then go immediately to Part A.
4. Do not read the example first as you'll be influenced to use the example and hardcode it. Free response questions must handle generic input values - not the example explicitly.
5. Work on one part of each question at a time.  If you attempt to read part A, B, and C, your mind will be filled with too many details.  “Divide and conquer” each question.
6. Rarely will you find a question that asks you to use the Scanner class or System.out.println() to answer the question.
7. Remember to make your class variables private.
8. Do not get so bogged down on one free response answer that you do not complete all four of them. You have ~22.5 minutes per question.
9. Do not redefine an existing instance variable.
10. Pay attention to the return type of the method
	* If void, do not return anything.
	* If it has a return type, start by declaring a local variable of that type and then returning that variable at the bottom of the method so that you won’t forget to do that.
11. Be sure to reuse the code that has either been provided to you or you have implemented in Part A of the question. Re-implementing any code may result in losing all the points for that question.
12. Use the Quick Reference guide (the one-page version).
13. All legal Java is legal and will earn you full points if correct. However, don’t write a fancy solution if a simple one will do. The exam is graded by humans and they can make mistakes.

**//Topic Reminders**

**Number Conversions**

* Make sure you know how to convert from a decimal number to a hex, octal, or binary number.

**Integers**

* Constants
	+ The maximum and minimum values of a primitive int variable are represented by the constants Integer.MAX\_VALUE and Integer.MIN\_VALUE. (These are on your reference sheet.)
	+ The value for the max is 231-1. You don’t have to memorize that, but you do need to know that if you add one to the max, it wraps around to the min.
* Remember integer division!
* You will not be tested on NumberFormat or DecimalFormat.
* Be able to extract a digit from a number.
* Be able to do modular arithmetic.

**Shortcut Assignment Operators and Symbols**

* Given int x = 5; understand the difference between:
	+ - x \*= x + 1; ---> 30
		- x = x \* x + 1; ---> 26
* If you want to multiply a and b, remember to write a\*b and not just ab.

**Escape Sequences**

* Understand escape sequences inside strings \\, \", \n.

**Short-Circuit Evaluation and DeMorgan’s Law**

* Remember conditions are evaluated from left to right and once the condition is known it stops executing.
* DeMorgan’s Law says that if you distribute a not, then you change the operator to its opposite.

**Random and Casting**

* int x = (int)Math.random() \* 5; ---> always 0
* int x = (int)(Math.random()\* 5); ---> some number
* Multiplication is used to increase the range and addition is used to change the offset
	+ int x = (int) (Math.random() \* range ) + lowestValue;

**Strings**

* substring (2,3) returns the character in index 2 but it does not include the character in index 3.
* The 2nd argument in substring(x, y) can include the length of the string without going out of bounds.
* Know what substring(x, x) returns.
* indexOf(String str) returns -1 if str is not found within the calling string.
* Know how to change an int, double, and char into a String – just add it to an empty set of quotes.
* str1.compareTo(str2) will return a -1 if str1 comes before str2 in lexicographic order (caps come before lower case letters)
* String str = new String (“test”) and String str = “test” are not the same thing when it comes to memory location.

**== vs. .equals**

* Use == for comparing primitives (ints, booleans, etc.)
* Use .equals for comparing objects (including Strings).
* For objects, .equals checks for value and == checks memory location.
* If you set one object equal to another (without actually instantiating the 2nd object), they point to the same space in memory.

**For-Each Loops**

* Can not be used to modify the elements of the array/ArrayList.

**Arrays**

* Use [ ] to access and assign elements to/from an array.
* Use .length to determine how long the array is.
* To print the contents of an array, write a loop to traverse all the elements.
* When an array is first created, it contains the default values for its data type even before you put values in it.
* Know how to:
* compare consecutive elements in an array
* accumulate the values in an array
* find the average of the values in an array
* find the maximum and/or minimum values of an array
* find the index of the maximum and/or minimum values of an array
* swap the elements of an array
* expand an array
* search for an element in an array using linear search
* sort an array using selection sort

**ArrayLists**

* Use get/set to access and assign elements to/from an ArrayList.
* Use .size() to determine how long the ArrayList is.
* Use generics when declaring an ArrayList.
* If you are going to remove something from an ArrayList, remember that ArrayLists shift everything up when something is removed. You will have to step your loop back to recheck the space you are on.
* To print the contents of an ArrayList, you can just print the name of the ArrayList.

**Maximum/Minimum**

* If you have to find the max or min of an array, initialize the max/min to be an element of the array.
* Do not use any magic numbers (yes, 0 is considered a magic number in this case.)

**2D Arrays**

* Rows are horizontal, columns are vertical.
* [0][0] is in the upper left hand corner.
* 2-D arrays are just an array of arrays.
* array.length returns the number of rows.
* array[0].length returns the number of columns in row 0.
* Be able to move around the 2D array in multiple directions.

**Searching**

* Linear and binary search will be on the test.
* Make sure you know how many cuts it takes to find an item among 10,000 items using binary search.

**Sorting**

* Selection, insertion, and merge sorts are testable topics.
* You will not have to write the code, but you should know how they work and be able to answer questions about best and worst case scenarios for each.

**Class Writing**

* Make all instance variables private.
* While all instance variables get auto-initialized to their default values, it is a good idea to give them initial values in the constructor (but don’t make these up, use the specifications in the problem.)
* Double check all the methods you write and make sure you returned something if necessary and it is the correct type.
* Pay attention to pre- and post-conditions.
	+ Pre-conditions are like little gifts…you don’t have to worry about checking for whatever the pre-condition guarantees you.
	+ Make sure to not violate a post-condition.
* Unless specifically stated in the question, methods should not have static in the header.
* While you can technically leave off public in the class header with no deduction, it must be included in the constructor header and on all public methods. So, just put it everywhere, unless you are specifically writing a private method.
* Variables declared inside a method (including constructors) are local variables who scope is defined to just that method.
* You can write helper methods, but please be sure to actually call upon that method.

**Method Overloading vs. Overriding**

* Overloading is when you have methods in the same class with the same name but different parameters (best example is what we did with constructors).
* Overriding is when you have methods in the parent and child classes with the same name and same parameters.
* The signature of a method depends only on the number, types, and order of its parameters but does not include the return type of the method.

**toString()**

* If you print an object, that is the same thing as calling the toString method on an object (the JVM knows to look for a toString() method).
* If a toString method in the object’s class is found, then it executes that method.
* If a toString method is not found, then it calls the Object class’ toString method which will print the class name along with the memory address of the object.

**Inheritance**

* Remember that the subclass’s constructor **ALWAYS** calls the superclass’s constructor first even if it is not specifically written.
* Use extends to use inheritance.

**Polymorphism**

* Given Pet curlySue = new Labradoodle ();
	+ At compile time, curlySue is a Pet and all methods to be executed on her must exist in Pet.
	+ At runtime, curlySue is a Labradoodle and the Java Virtual Machine looks in Labradoodle for methods first.
* To call a superclass’ method that has been overridden in the current class, you need to invoke the method by using super.methodName().

**Interfaces**

* Use implements to use interfaces.
* Interfaces can only contain abstract methods and public final static variables.
* When you implement an interface, you have to write all the methods in the interface and the headers have to match exactly (without the word abstract).
* Testable interfaces: List<E> and Comparable<E>.

**Recursion**

* Trace your code by writing all your recursive calls in an appropriate stack.
* Print statements before recursive calls get executed in the order in which you see them.
* Print statements after recursive calls get executed in reverse order.

**Exceptions and Errors**

* ArrayIndexOutOfBoundsException, ClassCastException, ConcurrentModificationException, IntegerOverflow,
* possible loss of precision
* NullPointerException – avoid by:
	+ ensuring objects are instantiated before use
	+ declaring instance variables outside of the constructor
	+ checking cells in arrays and ArrayLists for null before trying to access objects within

**Not Covered on Exam**

**Brainstorm a list of things you may covered that’s not testable on the exam. Write small – there is a lot.**