AP Computer Science A

Review for AP Exam

Exam: Wednesday, May 3rd, 2023 @ 12PM

Exam Layout

Section I: 40 multiple choice, 1 hr 30 min

Section II: 4 free response, 1 hr 30 min

- 1 Method and Control Structures question
- 1 Classes question
- 1 Array/ArrayList question
- 1 2D Array question

Materials

- ***NO*** calculator
- No. 2 Pencils
- Pens with blue and black ink
- A watch that doesn't make noise and doesn't have access to the internet (optional)

Exam Weight

- Primitive Types: 2.5-5%
- Using Objects: 5-7.5%
- Boolean Expressions and If Statements: 15-17.5%
- Iteration: 17.5-22.5%
- Writing Classes: 5-7.5%
- Arrays: 2.5-7.5%
- ArrayLists: 10-15%
- 2D Arrays: 7.5-10%
- Inheritance: 5-10%
- Recursion: 5-7.5%

Contents of Packet

There is a section for reference sheets
at the end of the packet. Please tear off for easy use.

NOTES

AP Exam → Review and Preparation

Concepts

- **De Morgan's Law**
  - Distributing the !
    - `!(x && y)` is the same as `!x || !y`
    - `!(x || y)` is the same as `!x && !y`
    - `!(x > 0)` is the same as `x <= 0`
    - `!(x < 0)` is the same as `x >= 0`
  - Shows how we can negate "and"s and "or"s
  - Proving De Morgan's Law logically with truth tables:

A	B	A&&B	A||B	!A	!B	!(A&&B)
T	T	T	T	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	F	T	T	T

- Random number with Math class
  - Use the following formula for a random *int*…
    - `int x = (int) (Math.random()*(max-min)+1)+ min;`
  - Use the following formula for a random *double*…
    - `double y = (Math.random()*(max-min)+1)+ min;`
- Short circuit evaluation
  - &&
    - If the first is false, the whole expression is false; there is no point to check the rest
    - Used to check null, prevent runtime errors, dividing by zero
    - `false && false → false`
  - ||
    - If first is true, the whole expression is true; there is no point to check the rest
- Compare methods
  - Compare To
    - `a.compareTo(b)` where `a` and `b` are Strings
    - Subtracts the lexicographic code of a and b as: a-b then returns said value

- Logical operators
    - Allows programs to make decisions based on multiple conditions
    - &&
        - If one is false, returns false
    - ||
        - If one is true, returns true
    - Boolean order of operations
        1) !
        2) &&
        3) ||
        - Order can be changed using parentheses
- Inheritance
    - All classes in Java inherit _____ from other classes
        - Attributes (instance variables)
        - Behaviors (methods)
- Constructors
    - If there are no constructors it will *not* error
        - If a class has no constructor in Java, the compiler will add a no-argument constructor
        - Creates a default empty constructor
    - Empty constructor
        - WILL NOT ERROR
        - A way to create an object without passing through specific parameters
        - Allows for the call `super();`
        - ALWAYS add an empty constructor when writing classes to avoid issues that may occur
    - Compile Time error
        - Occurs when a subclass has a no-argument constructor and variable is declared as `Class object = new SubClass();`

Types of Errors

- `NullPointerException`
    - When a method calls something that contains a `null` value
- Compile Time error
    - Code does not run, crashes before it can go through code
    - Syntax errors
- Runtime error
    - Goes through the program but crashes while in progress
    - Prints anything before it errors
    - Calling `.equals(null)` causes a run-time error

Sorting and Searching

- What to look for when deciding a sort/search…
  - Run-time efficiency
  - Size of the array
  - Amount of memory going to be used
- Searches
  - Sequential/Linear Search
    - Sequential way of searching through each elements until element is found
  - Binary Search
    - Examines the middle element and moves it left if element is less or right if element is greater
  - *Binary Searches are faster than Linear Searches*
- Sorting
  - Selection Sort
    - Selecting a value and putting it into its appropriate position in the list
    - Could swap a index value with greatest or smallest in array or list
  - Insertion Sort
    - Selects a value and compares it to the rest of the elements
    - Compares elements left of the selected value

Escape Sequences

- Allows certain actions in Strings
- Types
  - `\t` → Inserts a tab at the point of use
  - `\b` → Inserts a backspace at point of use
  - `\n` → Inserts a new line at point of use
  - `\r` → Inserts a carriage return in the text at the point of use
  - `\f` → Inserts a a form feed in the text at the point of use
  - `\'` → Inserts a single quotation (‘) at point of use
  - `\"` → Inserts a double quotation (“) at point of use
  - `\\` → Inserts a backslash (\) at point of use

Unit 1 → Primitive Types

Basics

- Syntax for a class and main method
  - `String [] args` → read the text as a string

```
public class MyClass
{
      public static void main (String [] args)
      {
            //code
      }
}
```

- Printing in the console
  - `System.out.println()`
    - Prints input *then* moves to the following line
  - `System.out.print()`
    - Prints input and *stays* on the *same* line
- Creating comments
  - `/* code here */`
    - Comments out a everything with in it
  - `/** code */`
    - Creates a bullet comment
  - `//code`
    - Comments out a singular line

Variables

- Variables
  - We name them using camelCase
    - Never starts with a number
    - Cannot contain any special characters *unless* it's an underscore (_)
  - Name associated with memory location in the computer
  - When you create a variable, you are declaring it
  - Stored as binary digits → 0 or 1
  - Using `final` means an `int` or `double` ***cannot*** be changed
  - Initializing a variable
    - FIrst mention of a variable
      - `int x; → x=0;`
      - `int x = 0;`
- Primitive data types
  - `int`
    - Stores integer values → {0,1,2,3,4…}
    - 32 bits → 2^{31}
  - `double`
    - Stores floating point numbers → {0, 1.1, 2.3, 3.14…}
    - Also known as a float
    - 64 bits
  - `boolean`
    - Stores true/false arguments → {true, false}
    - 1 bit
  - ***STRINGS ARE <u>NOT</u> PRIMITIVES***
- Strings
  - String literal
    - A string of text written with double quotes → " "
  - String concatenation
    - Use "+" to connect two or more strings

Variable declaration

- Assigning values
  - Variable being assigned a value *always* goes to the left side of the expression
  - Operators
    - Plus (+) → Adds variables together
    - Subtract (-) → Subtracts variables
    - Multiplication (*) → Multiplies variables
    - Division (/) → Divides variables
    - Modulus (%) → Takes the remainder of variables
    - Equals (=) → Sets a variable equation to an expression
    - Double equals (==) → Returns a boolean value depending on the expression
    - Not equal (!=) → Returns a boolean value depending if the expression is not equal to another
- Compound assignment operators

`+`	`-`	`*`	`/`	`%`
`x=x+1`	`x=x-1`	`x=x*1`	`x=x/1`	`x=x%1`
`x+=1`	`x-=1`	`x*=1`	`x/=1`	`x%=1`
`x++`	`x--`			

- Dividing with ints and doubles
  - int / int → truncates and cuts off the decimals
  - double / int → double
  - int / double → double
  - ((double) int/int) → double
  - (double) (int/int) → double but truncates because it divides ints first

Unit 2 → Objects

Understanding Objects

- Definitions
    - Class
        - A “blueprint”; always uppercase
    - Object
        - Instance of a class
    - Attributes
        - Objects properties
    - `private`
        - A method that can only be called in its class
    - Constructor
        - Allows information to be passed through a class
        - Used to create objects
    - Behavior
        - A type of method
    - Null
        - No value; empty variable
        - If not used correctly a `NullPointerException` will be thrown causing an error
- Getter and setter methods
    - Getter method
        - Gets, or returns, a value/variable
    - Setter method
        - Sets a value to a variable
        - Takes on parameters
- Behavior vs. attributes
    - Attribute
        - Variables and instance variable
    - Behavior
        - Methods in a class

Strings

- Understanding the syntax
    - String is capitalized because it is a class
- Strings are immutable
    - Immutable
        - Cannot be changed
    - Indices
        - The occurrence of a character
- String methods
    - int length()
        - Returns an int representing the how many characters are in a string
    - String substring(int start, int end)
        - Returns a subset of the string from the index passed through as parameters
    - int indexOf(String str)
        - Returns indice of imputed text
    - int compareTo(String other)
        - Returns a negative value if current string is less than the other
    - boolean equals(String other)
        - Returns boolean value if two strings are equals
    - String toUpperCase()
        - Turns every character in a string to uppercase
    - String toLowerCase()
        - Turns every character in a string to lowercase

Math Class

- A class used to use math functions
- A static class therefore we call using class name, not an object
- Methods
    - Math.random()
        - Returns a random double value
        - Has a range from $0 \leq x < 1$
            - $0 \rightarrow 0.9999$
        - Written as…
            - `(Math.random()*max-min+1) + min;`
    - Math.abs()
        - Returns the absolute value of a number
    - Other methods can be found on the Java documentation page
        - docs.oracle.com/javase/7/docs/api/

toString() Method

- Returns a string often containing variables related to the class

Unit 3 → Boolean and If Statements

Boolean Expressions

- Boolean expressions *always* return true/false
    - ==
        - Tests if two values are equal
    - !=
        - Tests if two values are *not* equal
    - <
        - Tests if one value is greater than another
    - <=
        - Tests if one value is greater than or equal to another
    - >
        - Tests if one value is less than another
    - >=
        - Tests if one value is less than or equal to another
    - We can use modulus to figure out if a number is even or odd
        - `x % 2 == 0` → even
        - `x % 2 != 0` → odd
- Introduction to De Morgan's Law
    - !(a && b) can also be written as !a || !b
    - !(a || b) can also be written as !a && !b

If Statement

- If an expression returns a certain boolean value, it will do certain lines of code
- Can be followed by an else or else if statement

```
if (boolean expression)
{
      //do this
}
```

- We can use while loops to do a repeated if statement until a certain boolean condition

```
while (boolean expression)
{
      //do this
}
```

String Equality and Null

- Strings can be written two different ways…
    - `String x = "hello world";`
    - `String x = new String("hello world");`
- Strings and boolean expressions
    - String x == String y
        - Tests if both variables point to the same place in memory
    - String x.equals(String y)
        - Tests if the variables contain the same information
- Working with null
    - String x;
        - Instantly stores a null value
    - If we use x for a method a `NullPointerException` will be thrown causing the code to error
        - Example…

            `x.indexOf("hello there");` prints…

            `java.lang.NullPointerException`

Unit 4 → Iteration

Loops

- While loop
  - A repeated if statement
  - Does code within until the boolean expression is *not* met

```
while (condition)
{
      //code
}
```

- For loop
  - Does a statement for a certain amount of time
  - Loops a definite amount of times and often known

```
for (int i = 0; condition with i; i++)
{
      //code
}
```

Unit 5 → Classes

Syntax of Classes

- Accessor methods
    - Getter methods
    - Allows you to access variables or values
- toString method
    - Returns a string description of a variable object
    - Converts an object to a string

Unit 6 → Arrays

Arrays

- Array is an object that can store many values of the same type in a single variable
    - Can be a list of `Strings`, `ints` etc.
    - Stores a ***fixed number*** of elements of the same type in a single variable
    - SIZE ***CANNOT*** BE CHANGED
- Data type
- Declaration
    - Syntax → `Type [] variableName = new type[numberOfvalues];`
        - `Type [] variable Name = {values};`
    - Example →`int [] score = new int[5];`
        - `int [] score = {1 , 2 , 3 , 4 , 5};`

Type	**Default Value**
int	0
double	0.0
boolean	false
Object	null

- Getting values
    - Enter index of values
    - `[0]` gives the *first* value
    - Example → `int whatScore = score[0];`
        - Prints out → `1`
- Getting length of array
    - Syntax → `arrayName.length;`
    - Example → `int scoreLength = score.length;`
        - Prints → `5`

Traversing Arrays

- Traversing an array is to cycle through an array using a loop
- Syntax

```
for (int i = 0; i < array.length; i++)
{
      System.out.println(array[i]);
}
```

- Iteration → amount of times the code runs
    - Usually equals to the `array.length` (during a `for` loop)
- `break;` stops the loop from continuing at the stop the line of code is

Enhanced For Loops

- An alternate method to transverse an array instead of using a `for` or `while` loop
    - Cannot exit while in-acton
- Also known as the For-Each Loop
- Efficient way to access objects
- Better used with nested loops
- Syntax

```
for (int variable : array)
{
      //code goes here
}
```

Developing Algorithms Using Arrays

- Common array algorithms
    - Max and min value
        - Minimum

```
int minIndex = 0;
for (int i = i; i < array.length; i++)
{
      if (array[i] < array[minIndex])
      {
            minIndex = i;
      }
}
```

    - Sum, average, or mode
        - For mode: use counter
        - Average

```
for (int i = 0; i < array.length; i++)
{
      //calculation here
}
return (double) sum / array.length;
```

    - Determining properties of a particular property
        - Properties of a value

```
int counter = 0;
for (int i = 0; i < array.length -1; i++)
{
      if (array[i].equals("property goes here")
      {
            counter++;
      }
}
```

    - Access consecutive pairs of elements

        - Check first number and if it is equal to the second, it's a consecutive pair
        - Consecutive

```
boolean consecutive = false;
for (int i = 0; i < array.length -1; i++)
{
      if (array[i] == array[i+1])
      {
            consecutive = true;
      }
}
```

- Reordering arrays
    - Shift or rotate elements left or right
    - Reverse order of elements
- Sorting arrays
    1. Start with a for loop on the first number
    2. Create a second for loop for the second number
    3. Take first number and compare to each number after
    4. If the first number is greater than the second, switch positions
    5. 2nd counter == 1st counter

Unit 7 → ArrayLists

ArrayList

- A class
- A mutable list of object references
- Syntax
    - `import java.util.ArrayList;`
    - `ArrayList <E> list = new ArrayList<E>();`
        - Where `<E>` can only be an object
        - Can also be declared as…
            - `ArrayList list = new ArrayList();`
    - `E` → represents the object being used
        - Excludes `int` and `double`
        - Use `Integer` and `Double`
- Array vs ArrayList
    - ArrayLists *are* changeable (mutable)
        - Can use `Integer`, `Double`, and `String`
    - Arrays are *not* changeable
        - Can use `int`, `double`, and `String`

ArrayList Methods

- Adding a number before the desire input places it in the numbered index
    - When added at a specific position, it shifts everything in the ArrayList right once
- `.size()` is used to find length of ArrayList
- Removing a number shifts everything in the ArrayList left once

Traversing ArrayLists

- `IndexOutOfBounds` → results from accessing index outside of range
- Can be accomplished with `while`, `for`, or `for` each loop
- Use `i--;` so it doesn't skip over an indice
- DO NOT USE AN ENHANCED FOR LOOP FOR ADDING AND REMOVING
    - Results in `ConcurrentModificationException`

Developing Algorithms Using ArrayLists

- `==` to check if lowercase or uppercase; `.equals()` to check for same exact
- `MyProgram.java: Line x: Out of memory! Please try again`
    - An error caused by an infinite loop

Linear Search

- We can use transversals to search for individual elements in an Array
- If it does not contain the target element, return -1
- Steps
    - Traverse through the ArrayList
    - Assignment the ith element to a variable
    - Set an If statement to see if variable is equal to target
        - If they are equal, return the index
    - Outside the for loop, return -1
- Linear (sequential) search checks each element until the target is reached
    - Could be used on `Integer`s, `String`s, or `Array`s
    - The longer the data size, the longer the process takes
- Don't use for each loop because they complete the loop regardless
    - Won't return the index

Sorting

- Organizing data can make it easier to search through

Selection Sort

- Sorts an array by repeatedly finding the minimum value and moving it to the front of the array
    - Implementation
        - For loop starting at 1 after the index
        - Traverse each `index` to the second to last element
        - Find minimum
        - Swap the `index` and `minIndex`

Insertion Sort

- Sorts an array by sorting each element compared to the elements already sorted to the left
- Implementation
    - Traverse each element starting at index 1
    - Shift sorted elements to place current elements
    - Start with a `for` loop where `index` (i) = `1`
    - CHECK EACH INDEX UNTIL CHOSEN VALUE HAS A LOWER VALUE THAN THE CURRENT INDEX
    - Create a `while` loop

```
while (sortedIndex > -1 && array[sortedIndex] >
array[currentIndex])
{
      array{sortedIndex+1]=array[sortedIndex];
      sortedIndex--;
}
array[sortedIndex+1]=currentIndexValue;
```

        - `sortedIndex` should be assigned to a value of `currentIndex-1`
        - `currentIndexValue` should be assigned to `array[index]`
    - **LAYOUT**

```
for (int i = 1; i < arr.length; i++)
{
      int curNumber = arr[i];
      int curIndex = i-1;
      while ( curIndex >= 0 && arr[curIndex] > curNumber)
      {
            arr[curIndex+1] = arr[curIndex];
            curIndex--;
      }
      arr[curIndex + 1] = curNumber;
}
```

Insertion vs. Selection

- Depends on how sorted the list is
- If somewhat sorted → *selection*
- If not at all → *insertion*

Unit 8 → 2D Arrays

2D Arrays

- Array that stores arrays
- *type* `[`*rows*`] [`*columns*`] array = new` *type*`[`*#rows*`][`*#columns*`]`
    - Rows → first bracket
    - Columns → second bracket
- Row Major Order → Process of traversing through a 2D array by moving one row into the next
- Column Major Order → Process of traversing through a 2D array by moving one column into the next
- Length of columns → `array[0].length`
- Length of rows → `array.length`

Traversing 2D Arrays

- Row major order
    - Traverse a 2D array across each row
    - Double for loop starting with row
- Column major order
    - Traverse a 2D array down each row, by the columns
    - `[`*column*`][`*row*`]`

Unit 9 → Inheritance

Inheritance

- EXAMPLE→ Person class
    - Instance variables of name and birthday (“has a” relationship)
    - A student is a person
- “has a” relationships help determine an instance variable
- “is a” relationships help determine class
- Superclasses
    - Extends parent class
    - Able to use methods from superclass in subclass without having to redeclare it
    - Syntax
        - `public class` SubClass `extends` SuperClass `{}`
    - `super()` must always come first in a constructor

Writing Constructors for the Subclass

- `super()`
    - Must *always* come first an a constructor
    - Passes parameters from a superclass to a subclass
    - Used to make a call to a superclass
        - If superclass doesn’t have anything, don’t include
- Subclasses *do not* inherit a constructor from the superclass
    - Subclass *must* have its own constructor
    - If there isn’t one present, Java will have a default one to be used

Overriding Methods

- Allows subclass to redefine a method instead of using a superclasses version
- Used when superclass and subclass has the same signature `public` class
- Override vs Overload
    - Overriding
        - In subclass and superclass
        - Same name, *same* parameters
        - `@Override` used to override a method
            - Not necessary, but good practice
            - Helps with debugging
            - Make code more readable
    - Overload
        - Same class
        - Same name, *different* parameters

`super` Keyword

- `super.`*`method`*`()` goes to the superclass and uses the method called there
    - Used to c;ass a superclass's method with correct parameters
- `super` is similar to `this`
    - `super` refers to classes
    - `this` returns to objects

Referencing

- Polymorphism is the capability of a method to do different things depending on the object it's acting on
    - Acts differently among classes
    - Poly=many & morph=forms
    - Take on different forms depending on implementation
- *SuperClass* `nameOfObject= new` *SubClass*`();`
- `ArrayList`s can have different subclasses of the same superclass

Unit 10 → Recursion

Recursion

- Definition
  - An iterative process where a method calls itself
  - Doing something repeatedly until forced out
- What does Recursion do?
  - Breaks a problem down into simpler sub-problems of the same for until it becomes easier to solve
  - Example
    - sum(10) = 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
      - sum(10) = 10 + sum(9)
- Base Case
  - A point where something cannot be simplified anymore
  - A way to exit a loop; get out
  - Not having a base case causes an infinite loop
  - Example

```
if (n==0)
      return 0;
```

or

```
if (n < 0)
      return 0;
```

- Formula
  - sum(n) = n + sum(n - 1)
    - Could be used as a return line
      - Returning a line that calls itself will cause the method to loop until it hits the base case
    - Similar to the arithmetic recursive formula
      - $a_n = a_{n-1} + d$
- Properties
  - Each recursive call has its own set of variables and parameters

- Implementations and traversing
  - Strings
  - Arrays
  - ArrayLists

Recursive Searching

- Review (from Unit 7)
    - Linear search checks each value in order until it reaches the end or desired value
- Binary Searches
    - Binary searches look for a midpoint and checks if value is greater or less than
    - ARRAY MUST BE SORTED
    - Very efficient because it eliminates half of the values
    - Walkthrough
        1) Test midpoint
        2) Eliminate half of population
        3) Find midpoint of remainder
        4) Repeat until target is found
    - Linear vs. Binary
        - Linear
            - Not efficient
            - Sorted or unsorted arrays
            - Easy to code
        - Binary
            - Very efficient
            - Sorted arrays ONLY
            - Slightly harder to code
        - Why?
            - Both offer the same efficiency from a memory and speed perspective
            - Personal preference and money
    - Formula
        - $number\ of\ iterations = log_2 array\ size$
            - $number\ of\ iterations = \frac{log(array\ length)}{log(2)}$
            - $array\ size = 2^{number\ of\ iterations}$
        - Maximum number of iterations needed for a binary search
- Encapsulation → trapped inside of itself

Recursive Sorting

- Merge Sort
    - Divides a list into 2 parts: left side and right side, then repeats this process until there is one number on each side. Finally it recombines them in order
    - Uses a recursive algorithm
    - More efficient than insertion and selection sort
        - Takes less time
        - Can be somewhat complex
    - Does *NOT* have to be in order
        - If there is an odd number of elements, right array would have more
    - Base case

```
if (length < 2)
      return;
```

    - Helper method → a private method used to simplify you code

Good luck on your exam!

Get that 5!

You got this!

Java Quick Reference

Accessible methods from the Java library that may be included in the exam

Class Constructors and Methods	Explanation
String Class	
`String(String str)`	Constructs a new `String` object that represents the same sequence of characters as `str`
`int length()`	Returns the number of characters in a `String` object
`String substring(int from, int to)`	Returns the substring beginning at index `from` and ending at index `to - 1`
`String substring(int from)`	Returns `substring(from, length())`
`int indexOf(String str)`	Returns the index of the first occurrence of `str`; returns `-1` if not found
`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`
Integer Class	
`Integer(int value)`	Constructs a new `Integer` object that represents the specified `int` value
`Integer.MIN_VALUE`	The minimum value represented by an `int` or `Integer`
`Integer.MAX_VALUE`	The maximum value represented by an `int` or `Integer`
`int intValue()`	Returns the value of this `Integer` as an `int`
Double Class	
`Double(double value)`	Constructs a new `Double` object that represents the specified `double` value
`double doubleValue()`	Returns the value of this `Double` as a `double`
Math Class	
`static int abs(int x)`	Returns the absolute value of an `int` value
`static double abs(double x)`	Returns the absolute value of a `double` value
`static double pow(double base, double exponent)`	Returns the value of the first parameter raised to the power of the second parameter
`static double sqrt(double x)`	Returns the positive square root of a `double` value
`static double random()`	Returns a `double` value greater than or equal to `0.0` and less than `1.0`
ArrayList Class	
`int size()`	Returns the number of elements in the list
`boolean add(E obj)`	Appends `obj` to end of list; returns `true`
`void add(int index, E obj)`	Inserts `obj` at position `index (0 <= index <= size)`, moving elements at position `index` and higher to the right (adds 1 to their indices) and adds 1 to size
`E get(int index)`	Returns the element at position `index` in the list
`E set(int index, E obj)`	Replaces the element at position `index` with `obj`; returns the element formerly at position `index`
`E remove(int index)`	Removes element from position `index`, moving elements at position `index + 1` and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position `index`
Object Class	
`boolean equals(Object other)`	
`String toString()`	