AP Computer Science A

Final Exam Study Guide

DOCUMENT CREATED BY YOU!!!!

Classes, Methods, and Definitions

Definition:

Central Processing Unit: performs program control and data processing.
Computer Program: tells the computer in minute details the sequence of steps needed to complete a task.
Hardware: The physical computer and peripheral devices
Software: The programs the computer executes
Programming: The act of designing and implementing computer programs
Compiler: translates the high-level instructions into the more detailed instructions (machine code)
Applets: Java code that can be located anywhere on the internet
Method: collection of programming instructions
Arguments: the technical term that methods need to carry out specific tasks
Run-time error (logic errors): syntactically correct and does something but not what it is supposed to do
Compile-time error (syntax errors): Wrong according to the rules of the language and the compiler finds it
Java Virtual Machine: Program that simulates a real CPU, a compiler that Java runs through
Boilerplate: Public static void main(string [] args) - declares the main method
Accessor method: a method that accesses an object and returns info without changing it
Mutator method: a method whose purpose is to modify internal data
Object reference: denotes the memory location of an object
Frame: window with a title bar
Component: what we want to show up inside the frame.
Cast: used to recover the Graphics2D object from Graphics of the paintcomponent method
The main method contains one or more instructions called statements.
Algorithm: sequence of steps that is unambiguous, executable, and terminated
Pseudocode: Informal description of the process to complete a task.
String: A sequence of characters enclosed within quotation marks
Integers: whole numbers
Variables: a storage location which has a name and holds a value
Public Interface: tells what you can do with objects of the class
Private Implementation: data inside its objects and the instructions for its methods
Package: Collection of classes with a related purpose
Test program: Verifies if one or more methods have been implemented correctly
Overloaded: Term which refers to names which have more than one method (ex: println method refers to 2 methods: println(int) and println(String))

Classes and Methods:

- Main Class
  - Main method: public static void main (String args[])
- Printstream Class
  - .println(String);
    - Prints out a given string or value and skips to the next line
    - Ex: System.out.println(String);
  - .print(String);

        - Prints out a given string or value without skipping to the next line
        - Ex: System.out.print(String);
- System Class
    - .out
        - Puts argument in the parentheses into the output box
    - .in
        - Takes input
- String Class
    - .length(String);
        - Returns length of string as an integer (Accessor)
    - .toUpperCase(String);
        - Changes string to all upper case (Mutator)
    - .toLowerCase(String);
        - Changes string to all lower case (Mutator)
    - .replace(String target, String replacement);
        - Replaces the portions of a string with a given replacement
        - arguments:
            - String target= the portion of the string which will be replaced;
            - String replacement = the string which will be inserted in every instance of the string target
- Rectangle Class
    - new: operator to be invoked when constructing objects
        - Example: new rectangle(int x,int y,int width,int height)
        - argument:
            - int x = initial x coordinate for rectangle;
            - int y = initial y coordinate for rectangle;
            - int width = initial width of the rectangle;
            - int height = initial height of the rectangle
    - .getx;
        - Gets x-value (accessor)
    - .gety;
        - Gets y-value (accessor)
    - .getwidth ;
        - Gets width of a rectangle (accessor)
    - .getheight;
        - Gets height of a rectangle (accessor)
    - .translate(x,y);
        - Moves the rectangle x units in the X direction and y units in the Y direction
    - .setSize(int1,int2);
        - Set the size of the Rectangle with a specified width & height
        - arguments:
            - int1 = rectangle width;
            - int2 = rectangle length

Other:

- API Documentation (Application Programming Interface)
  - Documentation lists all the classes and methods of Java library
  - https://docs.oracle.com/javase/7/docs/api/
- Variable Declaration
  - Example: type name = value;
    - Type: String, int, double etc.
    - Name = name of the storage location variable
    - value= what will be stored inside the variable

Chapter 3

3.1

- Each object of a class has its own set of instance variables.
  - Instance variables-a storage location that is present in each object of the class
  - To declare an instance variable:
    - an access specifier, the type of the instance variable, and the name of the instance variable.
    - Ex: Rectangle box = new Rectangle(5, 10, 20, 30);
      - Instance variables: x=5, y=10, width=20, height=30
  - The access specifier in the declaration of instance variables should be private.
- Encapsulation-process of hiding implementation details and providing methods for data access
  - Allows user to use a class without knowing its implementation, making it easier to alter the code

3.2

- When declaring methods, must provide method header and method body(statements that are executed when method is called)
- A method header consists of what parts?
  - Access specifier (public/private)
  - Whether it requires an object or not (static)
  - Return type (i.e: int, double, String) (use void if none)
  - Arguments (if any)
- Constructors set the initial data for objects
  - Have the same name as the class and have no return values
  - No arguments
- Public Interface-operations any programmer can use to create and manipulate the class's objects
  - Use documentation comments to describe the classes and public methods of your programs(javadoc)

3.3

- Class implementation
  a. Provide instance variables
  b. Provide constructors
  c. Provide methods
    - Accessor or mutator?
      - If mutator, make sure to label the correct return type in the method header

3.6

- Local variable - a variable that is declared in the body of the method.
    - Ex. Public double giveChange()
      {
      Double change = payment - purchase;
      Purchase = 0;
      Payment = 0;
      Return change;
- Parameter variables are similar to local variables, but they are declared in method headers.
    - Ex. public void receivePayment(double amount);
- You must initialize all local variables. Failure to do so will result in complaints from the compiler when you try to use it.
- Instance variable are initialized with a default value before a constructor is invoked.
    - Instance variables that are #s are initialized to 0.
    - Object references are set to a special value called null.
    - If an object reference is null, ten it refers to no object at all.
- Parameter variables are initialized when the method is called.
- Instance variables should be properly declared in the beginning of the class, right after the class is declared.
-

3.7

- This Reference
    - When implementing a method, provide a parameter variable for each argument
        - Parameter variables
            - Implicit parameters are not given by the user
            - Explicit parameters are given by the user
            - Ex: myAccount.deposit(500)
                - Implicit parameters: myAccount
                - Explicit parameters: 500
            - To access the implicit parameter, use the this reference
                - This reference is used as a placeholder for the identification of the object that is being invoked

Chapter 4

4.1

- Every value is reference to an object or belongs to one of the eight primitive types
    - Int - 4 bytes in size, range (-2,147,483,648 to 2,147,483,647)
    - byte - single byte numbers (-128 to 127)
    - short - short integer type, 2 bytes (-32,768 to 32,767)
    - long - The long integer type up to 9*10^18
    - double - double precision floating point type, 15 sig figs, +-10^308 range, 8 bytes
    - float - single precision floating point type, 7 sig figs, +-10^38 range, 4 bytes
    - char - character type (Unicode), 2 bytes
    - boolean - two true values (true or false), 1 bit
- Occasionally overflows when number literal is over the range of the primitive type
- It is okay to assign an integer value to a double, but it is not okay the other way around

- Floating point numbers, when divided or multiplied, might produce an error in the calculation because of the overflow of significant digits. To compensate, use rounding. If comparing doubles, it is best to create a constant called EPSILON (10^{-14} value) and see if the difference between the two numbers is less than EPSILON
- To declare constants, use 'final double' with the constant variable name in capital letters. Use underscores to indicate a space in the name
- When declaring a constant variable in a class use 'public static final double'.
    - static - indicates that constant belongs to class
    - final - indicates that the variable cannot be modified
    - public - can be accessed, safe because variable is already final
- Primitive types with fewer significant digits are more prone to roundoff errors

4.2

- Arithmetic Operators
    - ++ operator adds 1 to variable
    - -- operator subtracts 1 from variable
- If both numbers are integers, then the result of an integer division will be an integer, with the remainder discarded.
- To take the square root use Math.sqrt method
- For powers, use Math.pow(x, n)
- You can convert floating-point numbers to integers using the cast operator (int)
    - double balance = total + tax;
    - Int dollars = (int) balance;
- Rounding can be done through the Math.round method.

4.3

- Prompt - a message that tells the user which input is expected when the program asks for user input.

Ex. System.out.print("Please enter the number of bottles;");

- To read keyboard input, you use a class called Scanner. You obtain a Scanner object by using the following statement:
    - Scanner in = new Scanner(System.in);
- The nextInt method is used to read an integer value (This is only possible with the existence of a Scanner).
    - Example:
      System.out.print("Please enter the number of bottles: "):
      Int bottles = in.nextInt();
- Format Specifier - describes how a value should be formatted. (pg. 147, 4.3.2, Table 6)

4.5

- Literals - character sequences enclosed in quotes
- Number of characters in a string is the length of the string
- Empty String can be declared with " "
- Concatenate - combining two strings together to make one long string
- To include quotations use backslashes " He said \ "Hello\" "
- Strings are sequence of Unicode characters which are value type char.

- Each character has a numerical value
- The position of letters inside a string start at 0 and increase by 1.
- The substring method can be used to extract portions of a String
- The format for the substring method is: str.substring(start, pastEnd)

Chapter 5

The if Statement: Allows a program to carry out different actions depending on the nature of the data to be processed..

if (condition) {statement;} (could use else if, or else, or none at all)
else { statement2; }

Relational Operations: < (less than), <= (less than or equal), > (greater than), >= (greater than or equal to), == (equal to) , != (is not)

When comparing Strings, use the .equals method
ex) if (string1.equals.string2){ statement; }

Nested if Statements: When if statements are included within each other.
Boolean Variable: A variable that stores true or false.
Boolean Operators: && (only returns true if all statements are true)
|| (returns true if any one or more statements are true)
! ("not", returns the opposite true value)
In java, there cannot be compound equations; all of them have to be separate
(ex 1<x<2 REWRITE as 1<x, x<2)

Chapter 6

A loop executes instructions repeatedly while a condition is true.

Use while loops for conditions in which you don't know the number of iterations

While loops:
while(condition)
{
statements
}

For loops: Used when a value runs from a starting point to an ending point with a constant increment or decrement.

Use for loops for conditions in which you know the number of iterations

Example:
for (k = 0 (initialization), k < some number (condition), k++ (update))

```
{
        Statements
}
```

Do loops are appropriate when the loop body must be executed at least once. The condition is checked afterwards; thus being called a post-test loop.

```
do {
statements
}
while(condition)
```

Useful for input validation

Nested Loops: When the body of a loop contains another loop.

Common Loop Algorithms:

Sum and Average Value

```
double total = 0;
int count = 0;
while (in.hasNextDouble)
{
        double input = in.nextDouble();
        total = total + input;
        Count++;
}
if (count > 0)
{double average = total / count;}
```

Counting Matches

```
int spaces = 0;
for (int = 0; i < str.length;i++)
{
Char ch = str.charAt(i);
if (ch == ‘ ‘)
{
spaces ++;
}
}
```

Finding the First Match

```
boolean found = false;
char ch = ‘?’;
int position = 0;
while (!found && position < str.length())
```

```
{
        ch = str.charAt(position);
        if (ch == ' ') {found = truel; }
        else {position++; }
}
```

Prompting until a Match is found

```
Boolean valid = false
Double input = 0
While (valid) {
        System.out.println("Please enter positive value < 100");
        Input = in.nextDouble();
        If (0 < input && input < 100) {
        Valid = true;
        }
        Else {
        System.out.println("Number not valid");
}}
```

Maximum and Minimum Values

```
Double largest  in.nextDouble();
while (in.hasNextDouble())
{
        double input = in.nextDouble();
        if (input > largest)
        {
                largest = input;
        }
}
```

Comparing Adjacent Values

```
double input = 0;
while (in.hasNextDouble())
{
        double previous = input;
        input = in.nextDouble();
        if (input == previous)
        {System.out.println("Duplicate input.");}
}
```

Random Numbers and Simulations: use java.util.Random

- After constructing object of Random class, apply nextInt(n) or nextDouble()
- Ex: Random generator = new Random();
  Int d = 1 + generator.nextInt(6);
- Pseudorandom numbers are numbers drawn from very long sequences of numbers that don't repeat for a long time

For simulating a die roll

Monte Carlo Method: A method for finding approximate solutions to problems that can't be precisely solved

```
Import java.util.Random;
Public class Montecarlo
{
        Public static void main(String [ ] args){
        Final int TRIES = 10000;
        Random gen = new Random;
        Int hits = 0
        For (int i = 1; i <= TRIES; i++) {
                Double r - gen.nextDouble();
                Double y = -1 + 2 * r;
                If (x*x + y*y <= 1)
                {
                        hits ++;
                }
        }
        Double piEstimate = 4.0 * hits / TRIES;
        System.out.println("Estimate for pi: " + piEstimate);
        }
}
```

New methods:
in.hasNextDouble
string.charAt(index); returns the character at the specified index in a string
Chapter 7

7.1.1-

- An array collects a sequence of values of the same type
- Individual elements in an array are accessed by an integer index i using the notation *array*[i]
- An array element can be used like any variable
- Declaration and construction of an array:
  - dataType[] identifier = new dataType[i];
  - dataType[] identifier = {numbers};
- Access an element
  - identifier[i] = number;
- A bounds error which occurs if you supply an invalid array index, can cause your programs to terminate
- Use the expression array.length to find the number of elements in an array

7.1.2

- An array reference specifies the location of an array. Copying the reference yields a second reference to the same array.

7.1.3

- Provide a parameter variable for the array

Ex. public void addScore(int[] values)

```
{
  for(int i=0;i<values.length;i++)
  {
    totalScore+values[i];
  }
 }
```

Note that *int[] values* is a parameter variable that requires you to supply

- A method can also return an array

  Ex. public int[] getScore()

7.1.4

- Partially Filled Array is an array only partly occupied by actual elements. You must keep a companion variable that counts how many elements are actually in use.

```
Ex.   int currentSize=0;
  Scanner in= new Scanner(System.in);
  While (in.hasNextDouble())
 {
   if(currentSize<values.length)
   {
     values[currentSize]=in.nextDouble();
     currentSize;
   }
 }
 for(int i=0;i<currentSize;i++)
 {
   System.out.println(values[i]);
 }
```

7.2-

-
```
double[] values = …;
double total = 0;
for(double element : values)
{
  total = total + element;
}
```

  How to use enhanced for loop to total all elements in an array.

-
```
for (int i=0;i<values.length;i++)
{
 double element = values[i]
 total = total + element;
}
```

For loop with an explicit index variable

- ```
  for (double element : values)
  {
   element = 0;
  }
  ```

Gets elements of a collection

- ```
  for (int i = 0; i<values.length;i++)
  {
   values[i] =0;
  }
  ```

Sets element to 0

- Element variable is assigned values[0]
- Index variable is assigned values 0
- Enhanced for loop makes it easier to visit all elements in an array

7.3

- **7.3.1-Filling**
  - The following code fills an array
  - ```
    for (int i = 0; i< values.length; i++)
    {
            values[i] = i * i;
    }
    ```

- **7.3.2-Sum and Average Value**
  - ```
    double total = 0;
    for (double element : values)
    {
            total = total + element;
    }
    double average = 0;
    if (values.length > 0)
    {
            average = total / values.length;
    }
    ```

- **7.3.3-Maximum and Minimum**
  - ```
    if (int i = 1; i < values.length; i++)
    {
            if (values[i] values.largest)
            {
              largest = values[i];
            }
    }
    ```

- **7.3.4-Element Separators**
  - Elements are usually displayed with commas or vertical lines
    ```

- Ex.) 32 | 54 | 67.5 | 29 | 35
- for (int i = 0; i < values.length; i++)
  {
    if (i > 0)
      {
        System.out.print(“ | ”);
      }
      System.out.print(values[i]);
  }

- **7.3.5-Linear Search**
  - Find the position of the first element
  - int searchedValue = 100;
    int pos = 0;
    boolean found = false;
    while(pos < values.length && !found){
  if(values[pos] == searchedValue)
  {
    found = true;
  }
  else
  {
    pos++;
  }
}

- **7.3.6-Removing an Element**
  - currentSize - companion variable for tracking the number of elements in the array.
  - If the elements in the array are not in any particular order, simply overwrite the element to be removed with the last element of the array, then decrement the currentSize variable.

- **7.3.7-Inserting an Element**
  - if (currentSize < values.length)
    {
      currentSize++;
      values[currentSize - 1] = newElement;
    }
  - if (currentSize < values.length)
    {
      currentSize++;
      for (int i = currentSize - 1; i > pos; i--)
      {
        Values[i] = values [i-1];
      }

```
        Values[pos] = newElement;
    }
```

- **7.3.8-Swapping Elements**
  - double temp = values[i];
    values[i] = values[j];
  - Set values[i] to the saved value with
    values[j] = temps[j];
- **7.3.9-Copying Arrays**
  - double[] values = new double[6];
    // Fill array
    double [] prices = values;

-

7.6 - Two Dimensional Arrays

- **Two-dimensional array** (matrix) - An arrangement consisting of rows and columns of values.
  - **7.6.1 - Declaring Two-Dimensional Arrays**
    - Constructor:
      - *dataType* [] [] *name* = new *dataType*[*# of rows or variable containing int*] [*# of columns or variable containing int*]
      - Ex: int [] [] list = new int [7][8];
        - Creates a 2D array with data type *int* named *list*. This 2D array has *7 rows* and *8 columns*.
  - **7.6.2 - Accessing Elements**
    - Accessing a certain element:
      - dataType name = arrayName[row # or int variable][column # or int column]
      - Ex: int element1 = name[1][2];
        - Creates a variable named *element1* with data type *int*. This variable is whatever number that is in *row 1, column 2* in 2D array *name*.
    - Accessing all the elements in a 2D array:
      - Create a nested for loop.
      - Ex: for (int i = 0; i < *rows*; i++)
        {
        for (int j = 0; j < *columns*; j++)
        {
        Statement to process for every element
        }
        }
        - Processes all elements in an array within the stated rows and columns.
  - **7.6.3 - Locating Neighboring Elements**
    - Positions of other elements relative to one element:

[i - 1] [j - 1]	[i - 1] [j]	[i - 1] [j + 1]
[i] [j - 1]	[i] [j]	[i] [j + 1]
[i + 1] [j - 1]	[i + 1] [j]	[i + 1] [j + 1]

7.7- Array Lists

7.7.1 Declaring and Using Array Lists

ArrayList: stores data from a sequence of values whose size can change

- can grow/shrink
- cannot store primitive types
- ArrayList class supplies methods for common tasks, such as inserting and removing elements

Array List Declaration:

ArrayList<data type>name = new ArrayList<data type> ();

ArrayList = Variable type

Name = variable name

ArrayList (2) = An array list object of size 0

The ArrayList class is a generic class: ArraryList<Type> collects elements of the specified type

7.7.2 Using the enhanced for loop

Enhanced For Loop example:

```
for (int i = 0; i < names.size(); i++)
{
        String name = names.gt(i);
        System.out.println(name);
}
```

Regular loop equivalent example:

```
for (String name : names)
{
        System.out.println(name);
}
```

7.7.4 Wrappers and Auto-Boxing to collect numbers in array, you must use wrapper classes

7.7.5 Using Array Algorithms with Array Lists

For loops (Arrays)

```
Double largest = values[0];
For (int i = 1; i < values.length; i++)
{
        If (values[i] > largest)
        {
                Largest = values[i];
        }
```

```
}
```

For loops (ArrayLists)

```
Double largest = values.get(0);
For (int i = 1; i < values.size(); i++)
{
        If (values.get(i) > largest)
        {
                Largest = values.get(i);
        }
}
```

7.7.6 Storing Input Values in an Array List

```
ArrayList<DataType> identifier= new ArrayList<DataType>();
        while(in.hasNextDataType())
{
        identifer.add(in.hasNextDataType());
}
```

7.7.7 Removing Matches

- Use the remove method to remove an element

7.7.8 Choosing Between Array Lists and Array

- Array lists are easier to use than arrays.
- Arrays have better syntax for element access and initialization
    - If the size of a collection never changes, use an array
    - If you collect a long sequence of primitive type values and you are concerned about efficiency, use an array
    - otherwise, use an array list

Mutator Methods	add(element): adds an element to the end, increases its size add(position, element): adds an element at system.out.println(name): prints entire array list remove(position): removes element at position set(position, element): replaces current value at position with given element (0<position<element.size())
Accessor Methods	Size : returns number of elements in ArrayList get(position): returns element value at given position (0<position<element.size())

Magpie and Elevens Lab

What is Magpie?

Magpie is a small dynamically-typed programming language built around patterns, classes, and multimethods. It's a simple implementation of a chatbot.

Methods

trim - checks and removes spaces around a string; The trim() method removes whitespace from both ends of a string. Whitespace in this context is all the whitespace characters (space, tab, no-break space, etc.) and all the line terminator characters; Returns a copy of the string, with leading and trailing whitespace omitted.

```
System.out.println(Str.trim() );
```

indexOf - returns the index within the string object of the first occurrence of the specified value; case sensitive

```
str.indexOf(searchValue[, fromIndex])
```

compareTo - compares this String to another Object.

```
int compareTo(Object o)
```

Elevens Lab: design and the Object Oriented Principles that suggested that design

-Class structure: This reference
System.out.println
public String toString() {
Solution code
-Deck constructor — This constructor receives three arrays as parameters. The arrays contain the ranks, suits, and point values for each card in the deck. The constructor creates an ArrayList, and then creates the specified cards and adds them to the list.
------>ranks = {"A", "B", "C"}, suits = {"Giraffes", "Lions"}, and values = {2,1,6}, the constructor would create the following cards:

Methods

isEmpty - This method should return true when the size of the deck is 0; false otherwise.

```
public boolean isEmpty()
```

size - This method returns the number of cards in the deck that are left to be dealt.

deal - This method "deals" a card by removing a card from the deck and returning it, if there are any cards in the deck left to be dealt. It returns null if the deck is empty. There are several ways of accomplishing this task.

Here are two possible algorithms:

-> **Algorithm 1**: Because the cards are being held in an ArrayList, ->call the List method that removes an object at a specified index, and return that object.

-Removing the object from the end of the list would be more efficient,

-requires a separate “discard” list to keep track of the dealt cards.

-> **Algorithm 2:** decrement the size instance variable and then return the card at size.

- the size instance variable does double duty

-it determines which card to “deal” and it also represents how many cards in the deck are left to be dealt.

- boolean