AP COMPUTER SCIENCE A

UNIT 1 PRIMITIVE TYPES:

- Syntax → the correct way to type the code so that the program will run
- primitives → int, boolean, double
- reference → string, random
- constant → value does not change (keyword "final")
- modulus examples
  - 20%7 = 6
  - 100%20 = 0 → %2 of an even num = 0
  - 4/3 = 1 %2 of an odd num = 1
  - 24/6 = 4
  - 5.0/2.0 = 2.5
- typecasting examples
  - int a = 5; int b = 4;
  - double avg = (double) a/b;
  - ↳ manually rounding a decimal to nearest int:
    - double roundMe = 53.6
    - int result = (int)(roundMe + 0.5) → making it an int gets rid of the decimal

Array → .length
ArrayList → .size()
String → .length()

UNIT 2 USING OBJECTS:

- class → blueprint for constructing objects
- objects → created using the class that holds the blueprint of the object, contains any # of constructors, referred to as an instance of the class
- String variables are objects
- empty vs. null
  - empty: "", length of 0
  - null: no value → any attempt to execute a method on a null String will result in a run-time error NullPointerException
  - String myName = new String(); → empty string
  - String myName; → null (must be initialized to do smth wit)
- spaces count as characters
  - "ice cream" length = 9, indexs = 0-8
- String concatenation examples
  - String firstName = "Harry"
  - int swords = 8;
  - int brooms = 4;
  - String result = firstName + swords + brooms → Harry84
  - String result = firstName + (swords + brooms) → Harry12
  - String result = swords + brooms + firstName → 12Harry
- .equals() → method to compare 2 strings
- compareTo() → 0 - 2 strings are exactly the same
  - neg - first string comes before the second string
  - pos - first string comes after the second string
  - String string1 = "Hello"
  - String string2 = "Hello"
  - int result = string1.compareTo(string2) → 0
  - String string1 = "Hello"
  - String string2 = "Kitty"
  - int result = string1.compareTo(string2) → neg
  - int result = string2.compareTo(string1) → pos
- upper case letters come before lowercase
  - "Hello" comes before "hello"
- length() method → returns the number of characters in a string
  - String word = "lol"; word.length() → 3

UNIT 3 BOOLEAN + IF STATEMENTS:

- Condition → a comparison of 2 values using a relational operator
  - ! not
  - && and
  - || or
  - !(a&&b) = !a || !b
  - not(a and b) = not a or not b
  - not(a or b) = not a and not b

UNIT 4 ITERATION:

- Iteration → repeating a statement more than once
  - for loop → when you know how many times you want to do a set of instructions
    - the variable inside the for loop is not known after loop exits
  - while loop → can use a variable of any data type as long as it is declared before

UNIT 5 WRITING CLASSES:

- instance variables → the virtual attributes that describe an object of a class
  - "has a" relationship
- private instance variables → other classes do not have access to the data stored in the variable without asking the object to provide it
- void methods → methods that simply only perform some action, do not return a value
- return methods → return the data type defined in the method signature
- public method → other objects from other classes can use the method
- private methods → only accessible within the class
- toString() method → should usually be overriden, returns a meaningful representation of the attributes

objectReference.methodName();

- reference variable → holds the memory address of a soon to be created, doesn't hold the object itself
- aliasing → when 2 different object reference variables contain the same address of an object
- nullPointerException → occurs when you attempt to perform a method on a reference variable that is null
- overloaded methods: → have the same name and a different return type but can differ
  - different number of parameters
  - same number of parameters but atleast one is a different type
  - same exact parameters but in dif orders
- static variables → a single copy of the variable is created and shared among all objects at a class level
- static methods → can only call other static methods and can only use or modify static variables, cannot access or change the values of the instance variables or call non-static variables in the class

On the exam, all instance variables are private, accessor or mutator methods provided by the class ex: myCircle.radius = 5

UNIT 6 ARRAY:

- array → a complex data structure because it can store a list of primitives or objects
  - each element can be referenced by an index
  - the number of elements in an array will stay the same, can't be resized
  - can use .length to find the total elements in an array arr.length
    - dataType[] array = {"cat", "dog", "mouse"};
    - dataType[] array = new dataType[x];
- the valid index values for an array are 0 through 1 less than the # of elements

- indexOf() method → returns the index of a character in a String, -1 if character not found
- substring() method → used to extract a specific character or group of characters from a String
- StringIndexOutOfBoundsException → using an index that is greater than or equal to the length of a String, or a negative index
- math.abs(x) → absolute value
- math.sqrt(x) → square root
- math.pow(x,y) → raises number to specified exponent
- math.random() → returns a random double [0.0, 1)

generate a number from 4-20:

```
int result = (int) (math.random() * 17) +4;
```

generate a number between 0-12:

```
int result = (int)(math.random()*13);
```

UNIT 7 ARRAYLIST:

- ArrayList → a complex data structure that allows you to add or remove objects from a list, changes size automatically

```
ArrayList<dataType> name = new ArrayList<dataType>();
```

- size() → returns how many elements are in the ArrayList
- add(E object) → appends the object to the end of the list
- add(int index, E object) → inserts the object into the position index
- remove(int index) → deletes the object from the list that is at index, returns the object
- get(int index) → returns the object at index
- set(int index, E object) → replaces the object at index with object

UNIT 10 RECURSION:

- recursive method → a method that calls itself
- all recursive methods must have a base case and a recursive call
  ↳ continues until the base case is satisfied
- merge sort → uses recursion; repeatedly divides the numbers into 2 groups until it can't do it anymore. Next the algorithm merges the smaller groups together and sorts them as it joins them. the process repeats until all groups form one group
- binary search → starts at the middle of a sorted array and eliminates half of the array in each iteration until the desired value is found or all elements have been eliminated

RANDOM:

- (double)(3/4) = 0.0
- traverse an array → move through each slot in the array, one at a time
- for each loop → iterates through each of the elements in the array, temp variable takes on value for each element

```
for(dataType i : arrayName)
{
  do smth w/ i
}
```

UNIT 8 2D ARRAY:

- 2D Array → can be visualized as a rectangular grid made up of rows and columns

```
dataType[][] name = {{val1, val23, {val3, val433};
dataType[][] name = new dataType[x][x];
```

rows →
columns ↓

- arr.length → gets number of rows
- arr[0].length → gets number of columns

UNIT 9 INHERITANCE:

- Child classes inherit the instance variables and methods of the parent class
  ↳ the child class can only access the private instance variables of the parent by using the accessor and mutator methods from the parent class
- the instance variables for the parent and the instance variables for the child are two different sets of instance variables

"child class extends parent class"

- a class can extend at most one other class
- super() → calls the parent constructor, must be first line of code
- no parameter constructor makes a call to the parent's no-parameter constructor
- parameter constructor calls the parent's parameter constructor
- polymorphism → different child classes of the same parent class can act differently
- override → keep same method signature, change the return/output
- the reference variable can be the parent of a child object
- can call the parent class method by using super

```
super.eat()
```

- a parent can't perform any of the methods that are unique to a child class