**AP Computer Science Exam Condensed Study Guide**

**Comments**

/\* This is a long comment

That can wrap over multiple lines \*/

//This is another comment style

**Strings**

* A String is not a primitive data type. It is an object with embedded methods. Be familiar with the following methods:
	+ **length()** – Returns the integer lengths of the String (how many characters)
	+ **substring(int n)** – Returns a new String starting at index n. Ex: “computer”.substring(5) returns “ter”
	+ **substring (int a, int b)** – Returns a new String starting at index a going up to but not including index b. Ex: “computer”.substring(1,5) returns “ompu”
	+ **indexOf(String)** – Returns the integer index of where one String is found in another, or -1 Ex: “computer”.indexOf(“put”) returns 3, “monitor”.indexOf(“fox”) returns -1 (not found)
	+ **compareTo(String)** – Returns integer value based on comparing the two Strings. 0 is return if they have the same characters, a negative number if first String is less than second, and a positive number if the first String is greater than the second, based on the first character that is different, based on ASCII value.
* The substring method is overloaded. substring(int n), starts at index n and goes to the end of the String.
* Know that you can use the + operator to add Strings together (concantenation)
* Escape characters – Know \n, \t and \”
* Use .equals when comparing the text of two Strings, not ==. == asks if they are the same String.
* If you want to append ints to a String, make sure the String is give an initial value first, even if it is just “”.
* Strings are considered immutable (unchangeable). All that you can do it recreate a new version of it.

**Primitive Data Types**

* int – integer division, 32 bits, -2,147,483,648 to 2,147,483,647 is range, can’t store a double value
* double – floating point, can store an integer
* boolean – just true or false, flag = !flag
* Casting – Forces whatever follows to be converted - int x = (int) Math.PI; (x = 3)

**Mathematical Operators**

* PEMDAS
* % - Modulus – Returns the remainder, 7 % 5 = 2, 3 % 8 = 3, happens at same level as multiplication and division
* Incrementing a variable - x++, x+=1, x=x+1 (all are equivalent) y /= 3; y = y / 3; (both are equivalen)
* Remember your parentheses
* Math functions: Math.pow, Math.sqrt, Math.abs(int), Math.abs(double),
* Math.random() – returns values between 0 and 1.
* int x = (int) (Math.random() \* 12)+3; <- Random number from 3 to 14. (12 possible random values, starting at 3)

**IF Statements**

* Nested IF statements
* if…else statements require that one and only one condition will be performed. (if or else)
* if…else if…else if – Can have any number of else if. Once one condition is met, it stop checking the rest.
* Recognize boolean shortcuts with if statements; if(done) is the same as if(done==true)

**Truth Tables**

* Order of precedence is NOT (!), AND(&&) , OR (||)
* DeMorgan’s Law - !(a && b) = !a || !b also !(a || b) = !a && !b
* Use parentheses as you would in normal mathematical expressed. In other words, do parentheses first
* To save time on these, find a way to short circuit the expression. For example, if you determine the first half of an && (and) expression is false, you know the entire expression will be false regardless of what the second half value is. Ex: (a > b) && (!b|| (b&& !a) 🡨 If you determine that a <= b, you know this will return false.
* You can also short circuit with || (or) expressions. If one side is true, the entire expression will be true.

**Wrapper Classes**

* Integer and Double should be known
* The behave like int and double, but have built in methods and constants.
* Integer.MAX\_VALUE, Integer.MIN\_VALUE are constants to be aware of. If you are trying to find the largest number of an array, it’s best to set the value of the variable to Integer.MIN\_VALUE, rather than 0, in the event all numbers in the array are negative.
* Integer num = new Integer(10); Double num2 = new Double(3.1415);
* You can’t use primitive data types in ArrayLists, so if the need arises to save integer or floating point values in an ArrayList, declare them as type Integer or Double. Ex: ArrayList<Integer> nums = new ArrayList<Integer>();
* Like with all objects, you will use .equals to compare one Integer or Double object to another.

**Arrays**

* Key terms: index (position in the array, starting at 0), element (an individual value within the array)
* Arrays can be used for primitive data types or objects
* Declaring an array:
	+ int [ ] nums = new int [10];
	+ int [ ] nums = {1,5,7,8,9};
* Assigning values to an element in an array: nums[0]=11;
* The number of elements in an array is found using .length (Ex: nums.length)
* The last element in an array is at index: nums[nums.length -1];
* Typically will use a for loop to go through an array.
* Be aware of ArrayIndexOutOfBoundsException, which occurs when you specific an invalid index.
* 2D arrays are like a matrix. Imagine you have: int [ ] [ ] nums = new int[3][4];
	+ Number of rows is nums.length and is the first number. 3 rows in nums.
	+ Number of columns is the length of the first row or nums[0].length; 4 cols in nums.
	+ We use nested for loops to go through all elements in a 2D array

**Methods**

* Primitive data types sent into a method are unchanged outside the array. Only their value is sent in.
* Arrays and ArrayLists sent into a method are changed outside the method, if they are changed inside the method.
* Overloaded and Overriden Methods (See Object Orient Concepts)
* Abstract methods require subclasses to build them and only have the method header (no implementation)
	+ abstract

**Object Oriented Concepts**

* Inheritance
	+ Inheritance is when one class “is-a” type of another class.
		- Examples: Vehicle and Car classes, Shape and Square, Animal and Mammal and Tiger
	+ If a class doesn’t extend another class, it automatically extends the Object class.
		- All objects will inherit the equals method.
		- public boolean equals(Object o) – Returns true if o is the same the object calling equals.
		- Equals can be overridden, as it is in the String class.
	+ A class can only extend one other class.
	+ Superclass
		- Its data fields and methods can be received by other classes
		- “Has-a” is sometimes used to describe the superclass
	+ Subclass
		- Receives data fields and methods from a superclass
		- Declared with extends keyword. Ex: public class Triangle extends Shape
		- Constructors are never inherited!
		- super keyword calls the superclass method or constructor instead of its own.
* Association
	+ This is not inheritance, but one class is a data field of another class.
	+ For example, in Battleship, our Board class used a 2D array of Coordinate Objects, yet Board did not extend Coordinate. The Coordinate is simply used to help define the Board Class.
* Private Data Fields
	+ Any data fields declared as private will require getter (accessor) and setter(mutator) methods to be created so other classes can call upon and change these data fields.
* Static Data Fields
	+ The keyword static means “shared”. All instances of this object will share the value stored in this data field.
	+ If any instance changes the value stored in a static data field, it is changed for all instances.
* Polymorphism
	+ Overriding a method
	+ Allows there to be multiple versions of the same method, but each must have different parameters
	+ Downcasting
		- Imagine you had class Shape and class Triangle which extends Shape
			* Shape s = new Triangle(); //This is valid since all Triangles are Shapes.
			* Triangle t = new Shape(); //Invalid since all Shapes are not Triangles.
* Overloading vs. Overriding
	+ Overload means multiple method with the same name but different parameters.
	+ Override means recreating an inherited method at the subclass level.
* Abstract Class
	+ Contains abstract methods which every class that extends will have to construct
	+ Cannot instantiate an abstract class.
	+ Designed to be inherited
* Casting as an Object
* Interface
	+ Used to define behaviors
	+ A class can implement multiple interfaces
	+ Contains a group of unimplemented methods (no body, just header)
	+ Any class that implements (keyword) an interface must construct these methods.
	+ You can downcast with an interface as long as the class being instantiated implements the interface or inherits the class that implements the interface.
	+ Comparable interface
		- Requires object that implements it to create the compareTo method
		- public int compareTo(Object o)
		- Compares 2 Objects of some type and returns an integer representation of the comparison.
	+ Example:

public interface Edible(){

void taste();

public class MacIntosh extends Apple{

String color;

public MacIntosh(String color){

 this.color = color;

}

public void eat(){

 System.out.println(“Tart”);

}

String howToEat();

}

public class Apple implements Edible{

int calories;

public Apple(int calories){

this.calories = calories;

}

 public void eat(){

 System.out.println(“Sweet”);

 }

 public String howToEat()

 {

 return “bite by bite”;

 }

}

* Edible apple = new Apple(); Valid since an Apple has all of the method contained in the interface.
* Edible apple = new Edible(); Invalid since you cannot instantiate an interface
* Searches and Sorts – See Separate Document