APCS Cheat Sheet

Includes syntax reminders, and solutions to common things that might be on the exam. Obviously, it won't be the same question. But these might remind you how to do a question. Use ctrl / cmd F to search for something on this sheet that might be relevant.

Special thanks to my APCS teacher for compiling a list of the questions and terms on this cheat sheet!

Disclaimer: If you use this and do not pass please do not blame or sue me.

TREVOR PACKER AND THE COLLEGE BOARD THIS IS A PDF I CANNOT EDIT IT DURING THE EXAM PLEASE DO NOT ARREST ME.

Things to remember / random shit (in randomish order):

➔ When writing a method, if you have time, check for edge cases.

➔ How to initialize an object:

Object objectName = new Object(parameters);

For example:

Dog wilson = new Dog("wilson");

➔ Boolean logic:

isequalto == – true if are equal

or || – ANY of the conditions can be met

if(1==1 || 1==2) is true

and && – ALL of the conditions must be met

if(1==1 && 1==2) is false

not ! – turns true into false and vice versa

if(!(1==2)) is true

I'm not sure why you would have to know the following but:

You can factor out a negative and change the operator for the same result:

if(!(1==2) || (1==1)) is the same as if(!(1==2 && !1==1))

The logic is pretty nice but this is honestly probably not gonna come up

➔ You must use .equals() when comparing Strings.

➔ Anatomy of a header:

access (static) returnType methodName(paramType parameter,...)

Examples:

public static void sayHi();

public int timesTwo(int num);

public Dog findDogByName(String name);

➜ Variables in a class are usually private, methods in a class are usually public.

➜Java doesn't throw an error at you if you index / slice 1 slot too much in a String, so don't worry about an out of bounds error when you loop once too many times.

For example:

"wilson".substring(5,6) works properly, even though "wilson" technically doesn't have a 6th letter.

➜ Use void when you don't want to return anything

➜ You can use "return;" in a void method to kill the method

➜ Getting the size:

.length() for a String
.length for an array
.size() for an ArrayList

➜ How to initialize an array:

Object[] arrayName = new Object[desiredSize];

Or

Object[] arrayName = {object1, object2, ...}

For example:

int[] numberList = new int[3];
int[] numberList = {1,2,3};

➜ The compareTo() method:

String1.compareTo(String2) returns a positive, negative, or 0 number

positive if String1 is AFTER String2 in the alphabet

"z".compareTo("a") returns a positive integer

negative if String1 is BEFORE String2 in the alphabet

"ab".compareTo("ac") returns a negative integer

zero if String1 is equal to String2

"abc".compareTo("abc") returns 0

What is the positive integer?

You probably don't have to know this, but it's a numerical value for how far apart the two Strings are.

➜ How to swap two values in an array (using temp):

We have an array:

int[] numArray = {0,1,2,3};

We want to swap the items at index 0 and 2

```
int temp = numArray[0];
numArray[0] = numArray[2];
numArray[2] = temp;
```

➔ To generate random number:

int random = (int)(Math.random()*10)+1; //returns ran num from 1-10

To change the range:

Multiply by your desired range

(int)(Math.random()*50) returns a random number from 0-49

To change the starting / ending number:

Add your desired starting number

(int)(Math.random()*50)+4 returns a random number from 4-53

➔ To use / call a method if you are inside the class that the method belongs to:

methodName(parameter1, parameter2, ...)

For example:

```
int num = thisIsAMethodThatReturnsANum();
```

➔ To use / call a method if you are outside the class that the method belongs to:

objectName.methodName(parameter, parameter2, ...)

For example:

```
int num = obj.thisIsAMethodThatReturnsANum();
```

➔ How to get a single letter from a String, knowing the index of that letter:

stringName.substring(index, index+1);

➔ How to initialize an ArrayList:

ArrayList<Object> listName = new ArrayList<Object>();

For example:

ArrayList<Integer> intList = new ArrayList<Integer>();

You can put objects into ArrayLists, but not primitives (int, double). Use Integer to represent a number.

➔ How to for loop:

```
for(int i=0; i<max; i++) {
    //do something
}
```

➔ How to for each loop:

```
for(Object objectName: iterableName) {
    //do something
}
```

For example:

```
stringList is an ArrayList of Strings
for(String item: stringList) {
        System.out.println(item);
}
```

Use this if you don't need to modify the ArrayList / array for whatever reason, but you only need to access it.

➜ Retrieve an item from an array:

```
arrayName[index];
```

➜ Retrieve an item from an ArrayList:

```
arrayListName.get(index);
```

➜ To check if a substring is in a String, use stringName.indexOf(substring) – this will return -1 if the substring is not in the String stringName

➜ When working with ArrayLists, inserting or removing items will cause the objects inside to shift indexes, which may mess up your loops. Just watch out for this.
➜ If an object inside an ArrayList or array is null, you can check if it's null, but if you try to do something with it, it will throw an error at you.
Example:

```
ArrayList stringList is {"hi", null, "wilson}
if(stringList.get(1) == null) // returns true, is ok
if(stringList.get(1).equals("hi")) // throws an error because null does not have
.equals() method
To bypass this, do:
        if(stringList.get(1) != null && stringList.get(1).equals("hi"))
```

If it is null, java will not run the second part because the boolean has already failed, meaning you will not get an error if it is null

Key Terms (featuring my crappy explanations via google):

➜ **Accessor:** method that returns the value of a private variable in a class

➜ **Assertion:** used to declare booleans, probably not relevant on exam

➜ **Assignment:** literally assigning a value to a variable using =

➜ **Bytecode:** code that has been compiled so your software interpreter can run it

➜ **Cast:** making a variable behave like a variable of another type

➜ **Constant:** a variable that cannot have its value changed after it is declared (more like an invariable haha am i right)

➜ **Data abstraction:** basically hiding the data so users can't change it

➜ **Encapsulation:** hiding code that people don't care about– so they can use your method without being bothered by implementation

➜ **Fields:** a variable within a java class. for example, a Dog class might have a name variable

➜ **Final:** class that can't be extended

➜ **Immutable:** object that can't be edited after creation. for example, String is immutable. you can't actually change the contents of a String, you simply reassign the variable to a new String whenever you make changes

➜ **Instance:** object created of a particular class

➜ **Mutator:** method that allows a user to change the contents of a variable in the class

➜ **Overloading:** allowing a class to have multiple methods with the same name as long as the methods have different parameters

➜ **Operator:** stuff like ! = | & + - % etc

➜ **Signature:** the method name and its parameter list

Possible questions and concepts that might help:

I use ArrayList in all my examples, but to do the same with an array, simply index it using name[i] instead of name.get(i);

It's likely that one of these "skills" will appear on the exam, but of course the question will be disguised. If you can get a general gist of these following skills you'll be able to recognize them on the exam and can refer back to this if you need a little bit of help.

I tested most of these out, but I'm tired as shit so I can't guarantee everything is 100% right. Let me know if you find any errors.

Loop through an ArrayList and see if the object is equal to something.

Example: We have a method to check if how many Strings in an ArrayList, StringList, is equal to a String passed into the parameters:

```
public int countSame(String check) {
	int count = 0;
	for(int i=0; i<StringList.size(); i++) { //loops through every item in StringList
		if(StringList.get(i).equals(check) { //checks if the String and check String are equal
			count++;
		}
	}
	return(count);
}
```

Loop through an ArrayList and compare adjacent values.

Example: We have a method that checks and returns true if any adjacent Strings in an ArrayList, StringList, are equal. (Two of the same Strings one after another.)

```
public boolean adjacentStrings() {
	for(int i=1; i<StringList.size(); i++) {
		if(StringList.get(i).equals(StringList.get(i-1)) {
			return(true);
		}
	}
	return(false);
} //explanation: goes through every list item except the first one, and checks if it is equal to the list item before it. if the program finds one that is equal, it returns true. if it goes through every item and doesn't find any that are equal, it returns false. the loop starts at i=1 because it compares the second item (i=1) with the first one (i=0)
```

Looking through a list and removing an object that meets a criteria.

Example: We have an ArrayList of Dog objects. In the Dog class, you see that each Dog has a getBreed() method that returns a String of the Dog's breed. This method will remove each Dog in an ArrayList of Dogs, dogList, that are the same breed as the one specified in the parameter.

```
public void removeDogBreed(String breedToRemove) {
	for(int i=0;i<dogList.size();i++) {
		if(dogList.get(i).getBreed().equals(breedToRemove) {
			dogList.remove(i);
```

```
                    i--;
                }
            }
```
}//i-- is necessary, because when you remove a Dog, the list shifts, so you need to subtract one from i to make sure it doesn't skip checking any Dogs.

Finding the smallest /largest number value in an ArrayList.

Example: Let's say you have an ArrayList of Dogs, dogList. The class documentation shows that the Dog class has a method getAge() which returns the age of the Dog. We have a method that returns the Dog that is youngest. If there are two Dogs with the same age, return any.

```
public Dog getYoungestDog() {
        int youngestAge = dogList.get(0).getAge(); //record the age of the first Dog
        Dog youngestDog = dogList.get(0); //record the Dog object of the current youngest Dog
        for(int i=1; i<dogList.size();i++) { //check each Dog in the rest of the list
            if(dogList.get(i).getAge() < youngestAge) {//if Dog's age is < than youngestAge
                youngestAge = dogList.get(i).getAge(); //update youngestAge
                youngestDog = dogList.get(i);//update youngestDog
            }
        }
        return(youngestDog);
}
```

Question where you need to do something with the individual digits in a number.

Example: a method that checks if the number passed through a parameter is one of the digits of an int num. Assuming that the number passed in cannot be 0.

➔ If num is 1938 and the number passed in the parameter is 3, return true. If num is 1938 and the number passed in the parameter is 4, return false.

```
public boolean isIn(int checkDigit) {
        while(num>0) {
            if(num%10==checkDigit) {
                return(true);
            }
            num/=10;
        }
        return(false);
```

} //explanation: pretend our number is 1938. num%10 is equal to 8, since % gives the remainder, and when 1938 is divided by 10, the remainder is 8. IF num%10 is equal to checkDigit, return true. If it's not, divide 1938 by 10 and set that equal to num. Since both are ints, the program will return an answer without the remainder. 1938/10 = 193. this is your new num. Repeat the loop. If 193%10 is equal to checkDigit, return true. Repeat the process, until you divide your last digit and end up with 0. If you are here, nothing returned true, so checkDigit is not in num. Return false.

The CB loves asking questions about digits in a number. Although it may not be the same question as the example, understanding how to use % and / by 10 to check the digits will be useful.

Checking if numbers are in ascending order (or Strings):

For putting Strings into ascending order, you can use String.compareTo();

Example: We have a method that returns false if an ArrayList, numList, is in strictly ascending order, and false otherwise.

```
public boolean isAscending() {
	for(int i=1; i<numList.size(); i++) {
		if(numList.get(i)<numList.get(i-1)) {
			return(false);
		}
	}
	return(true);
} //explanation: loop through each item. if any of the items are smaller than the one before it, the ArrayList is not in ascending order, so return false. if you get through all the numbers without returning false, return true
```

Inserting a number in its appropriate place (or Strings):

Again, for a problem like this but with Strings, just use compareTo()

Example: We have a method that needs to insert a number into an ArrayList, numList, into its appropriate place. The ArrayList is sorted by ascending order, so we want to put the number, num, into the ArrayList to keep the ascending order.

```
public void insertInPlace(int num) {
	for(int i=0; i<numList.size();i++) {
		if(num<numList.get(i)) {
			numList.add(i,num);
```

```
                    return;
                }
            }
            numList.add(num);
        } //explanation: go through the list until you find a number that your num is smaller than, meaning your num belongs before it – the last numList.add() is to add the num in case your num is bigger than everything in the list
```

Remove a single character from a string:

Example: we have a method that will remove all instances of the String smallWord in a String word.

```
public void removeLetter(String smallWord) {
        int startInd = word.indexOf(smallWord);
        int l = smallWord.length();
        while(startIndex>-1) {
            word=word.substring(0,startInd) +  word.substring(startInd+l);
            startInd = word.indexOf(smallWord);
        }
} //explanation, we find the index of a word, and remove it from a String until it no longer exists. note: this collapses a word, which may or may not be what you want
```

How to reverse a String:

Example: We have a method that will take a String and return a reversed version.

```
public String reverse(String reverseMe) {
        String reversed = "";
        for(int i=reverseMe.length()-1; i>=0; i--) {
            reversed = reversed + reverseMe.substring(i, i+1);
        }
        return(reversed);
} //explanation: loop through your string but from the end to the front, and add letters as you go
```

How to check if a String contains numbers or vowels or symbols or anything you want:

Example: We have a method that returns true if our String stringCheck has numbers in it and false if not. For example, "wilson" would return false, and "wilson69" would return true.

```
public boolean hasNum(String stringCheck) {
```

```
        String numbers = “0123456789”;
        for(int i=0; i<stringCheck.length(); i++) {
                String letter = stringCheck.substring(i,i+1);
                if(numbers.indexOf(letter)>-1) {
                        return(true);
                }
        }
        return(false);
} //explanation: loop through every character in stringCheck. check if the letter is contained in the String “0123456789”. if it is, then that “letter” is a number. return true. if we finish checking and that doesn’t happen, return false.
```