Java Cheat Sheet(s)

String Functions

- str.**indexOf(**String str, int fromIndex**)**
  - Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
- str.**indexOf(**substr**)**
  - Returns the index within this string of the first occurrence of the specified substring.
- str.**indexOf(**int ch, int fromIndex**)**
  - Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- str.**lastIndexOf(**int ch**)**
  - Returns the index within this string of the last occurrence of the specified character.
- str.**lastIndexOf(**int ch, int fromIndex**)**
  - Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
- str.**lastIndexOf(**substr**)**
  - Returns the index within this string of the last occurrence of the specified substring.
- str.**lastIndexOf(**String str, int fromIndex**)**
  - Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index
- str.**substring(**int beginIndex**)]**
  - Returns a new string that is a substring of this string.
- str.**substring(**int beginIndex, int endIndex**)**
  - Returns a new string that is a substring of this string.
- str.**charAt(**x**)**
  - Returns the char value at the specified index.
- str.**contains(**substr**)**
  - Returns true if the specified substr exists in a string
- str.**startsWith(**substr**)**
  - Returns true if the specified substr is a prefix of a string
- str.**startsWith(**String substr, int fromIndex**)**
  - Returns true if the String begins with str, it starts looking from the specified index “fromIndex”
- str.**endsWith(**substr**)**
  - Returns true if the specified substr is a suffix of a string
- str.**split(**x**)**
  - Returns an array of values derived from the string when it gets split at the specified regex i.e.
    String string = "004-034556"; *This would give us [“004”, “034556”]*
    String[] parts = string.split("-");

Java Cheat Sheet(s)

Inheritance

- *Subclasses (undergrad, postgrad) inherit from superclasses (student)*
- *A subclass is a specialisation of a superclass. A superclass is a generalisation of a subclass*
- *A subclass inherits all the members (instance variables and methods, public and protected) of the superclass*
- *In Java, a subclass extends a superclass*
- *Subclass can add new members of its own*
- *By default, methods that are inherited from a superclass have the same implementation in a subclass (unless a subclass overrides the inherited methods)*
- *Method m in subclass B overrides method m' in superclass A if m has exactly the same signature (i.e. name and parameters) as m'.*
- *Normally, m replaces the implementation of m'*
- *private instance variables (fields) cannot be directly accessed by subclass. They can only be accessed via setter and getter methods (which are inherited from superclass) i.e. superclass.getName()*

Functions:-

- **public class** subclass **extends** superclass
  - Creates a subclass which is an extension of an existing superclass
- **super.**method()
  - Calls the superclass's version of a method that you have overridden.
- **super(**parameter list**)**
  - Calls the specified superclass constructor(s)

Character Functions

- **Character.toUpperCase(**char**)**
  - Returns the lowercase version of the given character
- **Character.toLowerCase(**char**)**
  - Returns the lowercase version of the given character

Java Cheat Sheet(s)

Array List

- *Ensure you import java.util.ArrayList or java.util.**
- *Use this when you want your arrays to be able to grow, or you want to be able to easily insert or remove items in the middle of an array*
- *Unlike regular arrays, array lists can grow and shrink as needed.*
- *A newly constructed array list has size 0*
- *The size of an array list changes after each addition/removal*

Functions:-

- **ArrayList<** Type **>** Name **= new ArrayList<**Type**>()**
  - Declares a new arraylist with size 0
- Name**.size()**
  - Returns the size (length) of the arraylist in terms of an int
- Name**.add(**Item**)**
  - Adds an item to an arraylist
- Name**.get(**index**)**
  - Returns an item in an arraylist at a specific index
- Name**.add(**index**,** Item**)**
  - Inserts an item in an arraylist at a specific index
- Name**.clear()**
  - Clears an arraylist
- Name**.clone()**
  - Creates another copy of an existing arraylist
- Name**.contains(**Item**)**
  - Returns true if the specified item exists in an arraylist
- Name**.indexOf(**Item**)**
  - Returns the index of the specified item
- Name**.lastIndexOf(**Item**)**
  - Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
- Name**.remove(**Item**)**
  - Removes the specified item in an arraylist
- ArrayList2**.remove(**ArrayList1**)**
  - Removes all the items from the second list if it exists in the first list
- Name**.remove(**int fromIndex, int toIndex**)**
  - Removes all the items from this arraylist, whos index is between fromIndex and toIndex
- Name**.subList(**int fromIndex, int toIndex**)**
  - Creates a new arraylist with the values from an existing arraylist within the given range
- Name**.toArray()**
  - Turns an arraylist into a fixed array

Java Cheat Sheet(s)

HashMaps

- *Ensure you import java.util.HashMap or java.util.**
- *Use this when you want to use keys other than a predetermined list of integers*
- *Associative arrays (HashMaps) are a collection of unique keys with values*
- *In ordinary arrays, keys can only be integers (think the index of an array)*
- *Associative arrays allow keys of many types (i.e. strings, etc)*
- *Keys must be unique (just like with a regular array index [0..])*
- *A given key can only be mapped to one value*
- *The value can be any kind of object (i.e. an array)*
- *HashMap only takes 2 type parameters (key and value)*

Functions:-

- **HashMap<** KeyType, ValueType **>** Name **= new HashMap<**KeyType, ValueType**>()**
  - Declares a new hashmap with size 0
- Name**.size()**
  - Returns the number of key-value mappings contained in the hashmap
- Name**.put(**key**,**value**)**
  - Adds an item to the hashmap with the given key and value
- Name**.get(**key**)**
  - Returns an item in an hashmap with this specific key or null if no such key exists
- Name**.keySet()**
  - Returns the set of keys in a hashmap as a Set of keys
- Name**.clear()**
  - Removes all the mappings from the hashmap
- Name**.clone()**
  - Creates another copy of an existing hashmap
- Name**.containsKey(**key**)**
  - Returns true if the specified key exists within the hashmap
- Name**.containsValue(**value**)**
  - Returns true if the specified value exists within the hashmap
- Map2**.putAll(**Map1**)**
  - Puts all of the values in Map1 into Map2
- Name**.entrySet()**
  - Returns the hashmap as a Set of values i.e

    Set set **=** newmap**.entrySet();**

    Output: [1=tutorials, 2=point, 3=is best, key=value,…]

- Name**.remove(**key**)**
  - Removes the mapping for the specified key from this map if present.

Java Cheat Sheet(s)

HashSets

- *Ensure you import java.util.HashSet or java.util.**
- *HashSet doesn't maintain any order, the elements would be returned in any random order.*
- *HashSet doesn't allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.*
- *HashSet allows null values however if you insert more than one nulls it would still return only one null value.*
- *HashSet is non-synchronized.*

Functions:-

- **HashSet<** Type **>** Name **= new HashSet<**Type**>()**
  - Declares a new hashset with size 0
- Name**.add(**Item**)**
  - Adds an item to the hashset
- Name**.size()**
  - Returns the number of items stored in the set
- Name**.clear()**
  - Removes every element in the set
- Name**.clone()**
  - Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
- Name**.contains(**Item**)**
  - Returns true if the specified item exists in the set
- ArrayList**.remove(**item**)**
  - Removes the specified item from the set

Mathematical Functions

- **Math.min(**int a, int b**)** and **Math.max(**int a, int b**)**
  - Returns the smallest and largest value between two integers.
- **Math.abs(**x**)**
  - Returns the absolute value of the number (x) given.
- **Math.sqrt(**x**)**
  - Returns the square root of the number (x) given.
- **Math.atan2(**x**)**
  - Returns the arctan of the number (x) given
- **Math.PI**
  - Returns PI as a double constant

Java Cheat Sheet(s)

Miscellaneous

Comparing Values:-

- When comparing string values, use **.equals()** i.e. a**.equals(**b**)**
- When checking to see if two hashmaps are equal, first get their entry sets, compare their sizes and finally use Set1**.containsAll(**Set2**)** (Which returns true if set2 contains all the same values as set1)

Printing ArrayLists:-

- When printing an arraylist, use normal printing methods i.e.
  System.out.println(arraylistname**) == [**value1, value 2, value 3**]**

Printing HashMaps:-

- When printing a hashmap, use normal printing methods to get them printed in the format {Key1=Value1, Key2=Value2,…]

i.e. **System.out.println(**wordLengths**) == {**of=2, record=6, time=4, is=2, the=3**}**

Looping Over Arraylists:-

- When using a standard For Loop, use **.get()** i.e.

```
for (int n = 0; n < arraylistname.size(); n++){
        System.out.println(arraylistname.get(n));

}
for (Type s : arraylistname){
        System.out.println(s);

}
for (Type s : arraylistname){
        System.out.println(s);
}
```

Printing:-

- Use **\t** when printing a tab in a line i.e

```
for (Type s : cheers){
        System.out.print(s + "\thas index: ");
        System.out.println(cheers.indexOf(s));

}
```

```
hop     has index: 0
hip     has index: 1
hooray has index: 2
```

output

Java Cheat Sheet(s)

- Use **printf** and **%s** when printing out variables into a formatted line i.e.

```
for (String key : wordLengths.keySet()){
        System.out.printf("%s => %s" , key, wordLengths.get(key));
}
```

For the same purposes, this can also be used:

```
String output = String.format("%s = %d", "joe", 35)
```

- Use **\n** or **println()** when printing out a new line

Checking if an arraylist/hashmap/hashset/string is empty:-

- Use **isEmpty()** I.e. a.**isEmpty()**

When to use @Override

- Use this when you are writing multiple versions of the same function (i.e. if you have a function toString() in both the superclass and the subclass, you would override the method in the subclass if you change the output in anyway.

Access Modifiers

- **private** *<method>*
  - Can only be accessed within the declared class itself
- **public** *<method>*
  - Can be accessed from any class. Only needs to be imported if we're trying to access it from a different package.
- **void** *<method>*
  - The method has no return value
- **final** *<method>*
  - Once a final variable has been assigned, it always contains the same value. This means the variable is now immutable.