Java Cheat Sheet

Collections

Sorted Set **SortedSet<T> c = new TreeSet<();**	**Set** **Set<T> c = new**	**List** **List<Type> c = new ArrayList<>();**	**Map** **Map<Type> c = new Hashtable<>();**
TreeSet	*HashSet*	*ArrayList*	*Hashtable / HashMap*
These don't have a get() option, you have to iterate through them either with an iterable or an enhanced for loop		get(i) - returns value at given index;	put(key, new Value)- either creates new entry or replaces old value for corresponding key Get(key)- returns null if corresponding key doesn't exist, or the appropriate value corresponding to the given key keySet() - returns set of keys values() - returns collection of values

Where s1 and s2 are any appropriate collection objects you can use:
s1.containsAll(s2) — returns true if s2 is a subset of s1. (s2 is a subset of s1 if set s1 contains all of the elements in s2.)
 	
s1.addAll(s2) — transforms s1 into the union of s1 and s2. (The union of two sets is the set containing all of the elements contained in either set.)
 	
s1.retainAll(s2) — transforms s1 into the intersection of s1 and s2. (The intersection of two sets is the set containing only the elements common to both sets.)
 	
s1.removeAll(s2) — transforms s1 into the (asymmetric) set difference of s1 and s2. (For example, the set difference of s1 minus s2 is the set containing all of the elements found in s1 but not in s2.)
 	
S1.forEach(lambdaExpression) - performs an operation on all members of the collection
e.g. s1.forEach((n) -> System.out.println(n));
 	
S1.removeIf(lambdaExpression) - removes elements satisfying predicate
e.g. s1.removeIf((n) -> (n == 0));
 	
S1.toArray(T[])- converts the collection to given array type (DOESN'T WORK FOR PRIMITIVES)
e.g. Integer[] ints = s1.toArray(new Integer[0]);
+ANY METHODS FROM THE COLLECTIONS INTERFACE			

Switching between collection types

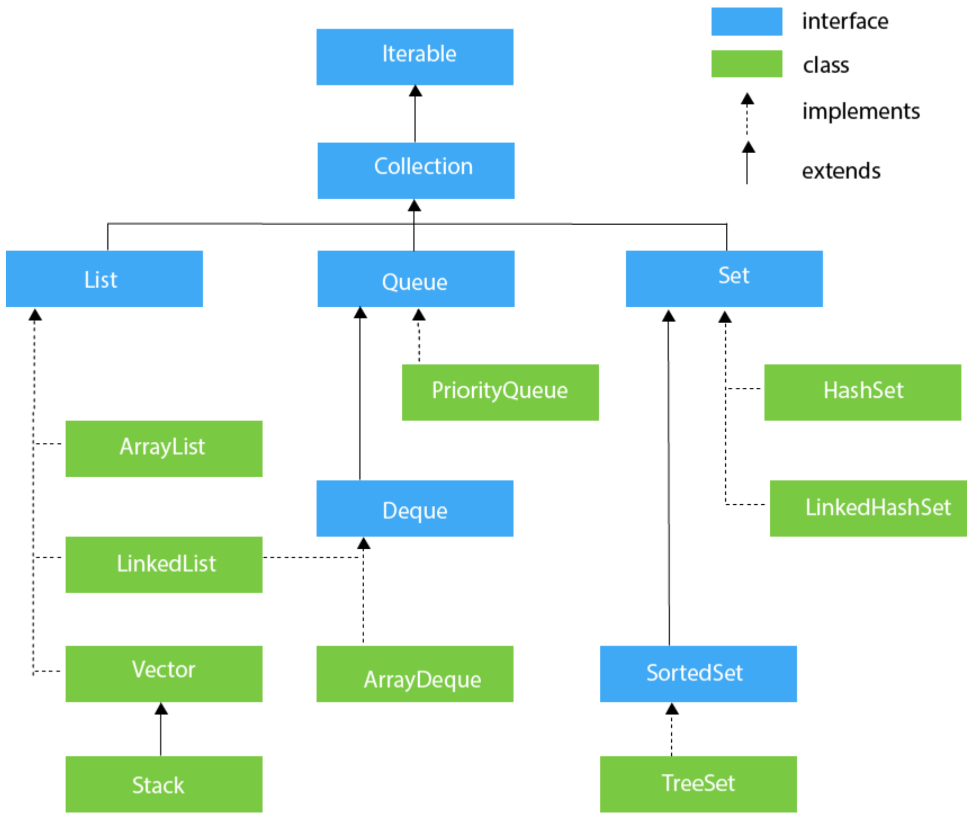
Suppose you have a collection **c** (could be ArrayList, a Set anything) then to swap to any other collection type you could for example:

List<Integer> newList = new ArrayList<Integer>(**c**);

SortedSet<Integer> newSet = new TreeSet<Integer>(**c**);

Also you can use **c**.toArray(T[]) to convert to an array (but only for object types e.g. Integer not int)

This is useful for sorting and removing duplicates as you can convert an array list to a set or ordered set and then back.



Sorting Collections

SortedSet does that for you, or if you need use the following:

For descending order:

Arrays.sort(**c**,Collections.reverseOrder());

Collections.sort(**c**, Collections.reverseOrder());

For ascending remove the second argument from the function call.

Reverse an ArrayList

Collections.reverse(**c**);

Reverse other collections (except hashmap)

Convert to array list with

List<T> myList = new ArrayList<T>();

myList.addAll**(c)**

Then reverse the arraylist and if you want convert back to old type

e.g. Set<T> oldReversed = new HashSet<T>(myList);

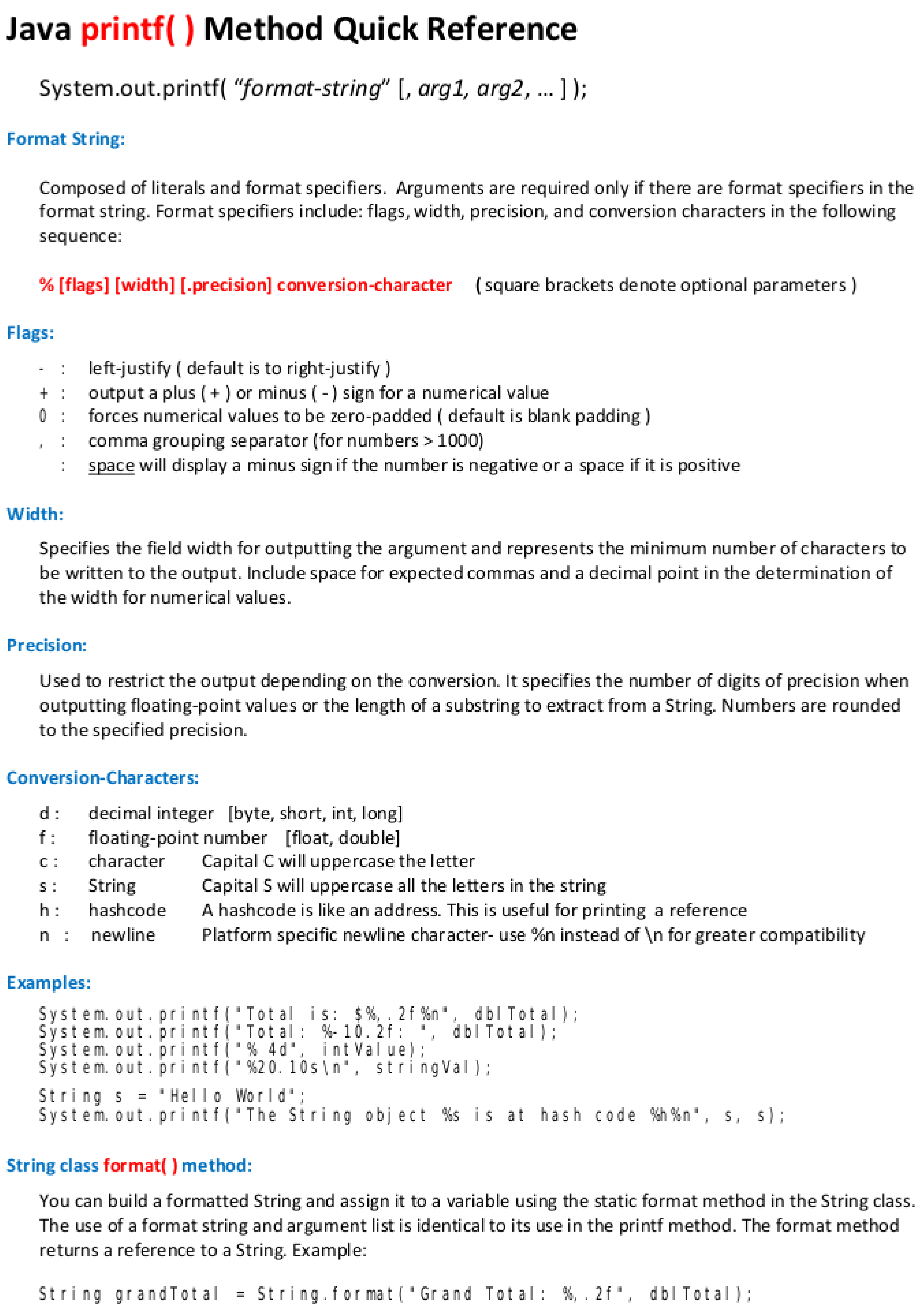
For each on Hashmaps

hm.forEach((k,v) -> System.out.println("key: "+k+" value:"+v));

TO FIND OTHER USEFUL FUNCTIONS, LOOK FOR THE CORRESPONDING INTERFACES IN THE DOCUMENTATION

Java Cheat Sheet

Printf()



Java printf( ) Method Quick Reference

System.out.printf(“*format-string*” [, *arg1, arg2*, …]);

Format String:

Composed of literals and format specifiers. Arguments are required only if there are format specifiers in the format string. Format specifiers include: flags, width, precision, and conversion characters in the following sequence:

% [flags] [width] [.precision] conversion-character (square brackets denote optional parameters)

Flags:

- :	left-justify (default is to right-justify)
+ :	output a plus (+) or minus (-) sign for a numerical value
0 :	forces numerical values to be zero-padded (default is blank padding)
, :	comma grouping separator (for numbers > 1000)
:	space will display a minus sign if the number is negative or a space if it is positive

Width:

Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output. Include space for expected commas and a decimal point in the determination of the width for numerical values.

Precision:

Used to restrict the output depending on the conversion. It specifies the number of digits of precision when outputting floating-point values or the length of a substring to extract from a String. Numbers are rounded to the specified precision.

Conversion-Characters:

d :	decimal integer [byte, short, int, long]	
f :	floating-point number [float, double]	
c :	character	Capital C will uppercase the letter
s :	String	Capital S will uppercase all the letters in the string
h :	hashcode	A hashcode is like an address. This is useful for printing a reference
n :	newline	Platform specific newline character- use %n instead of \n for greater compatibility

Examples:

```
System.out.printf("Total is: $%,.2f%n", dblTotal);
System.out.printf("Total: %-10.2f: ", dblTotal);
System.out.printf("% 4d", intValue);
System.out.printf("%20.10s\n", stringVal);

String s = "Hello World";
System.out.printf("The String object %s is at hash code %h%n", s, s);
```

String class format( ) method:

You can build a formatted String and assign it to a variable using the static format method in the String class. The use of a format string and argument list is identical to its use in the printf method. The format method returns a reference to a String. Example:

```
String grandTotal = String.format("Grand Total: %,.2f", dblTotal);
```

This is it I am afraid, remember no-compil-erino you succ-erino the big zero. -Maks :*