Java Tips

* \t Insert a tab in the text at this point.
* \b Insert a backspace in the text at this point.
* \n Insert a newline in the text at this point.
* \r Insert a carriage return in the text at this point.
* \f Insert a formfeed in the text at this point.
* \' Insert a single quote character in the text at this point.
* \" Insert a double quote character in the text at this point.
* \\ Insert a backslash character in the text at this point.
* Indent Vscode = Shift + Alt + F
* Create new Java Project in Vscode = Ctrl + Shift + P
* Make constant using “public static final”. E.g. final int pi = 3.16
* Close scanner once finished. E.g. input.close();
* When returning from void function use return; or nothing.
* Use Arrays.toString() to print our array contents.
* intArray.length gives length of intArray
* Arrays start with 0.

## Arraylists

* Arraylists are dynamic, don’t have to declare size of it beforehand, just an initial capacity if needed, or no size.
* ArrayList has a set of methods to access elements and modify them.
* ArrayList<Object> arrL = new ArrayList<>();
* Add elements – arrL.add(20);
* Access elements in index – arrL.get(index);
* Remove an element in an index – arrL.remove(index);
  + Element in next index moved into slot of removed index and size of arraylist decreased.
* Set an element at an index – arrL.set(index,value);
* Can only set elements to indexes that already contain element. Not to empty ones.
* Size of arraylist is found by arrL.size()
* for(Integer x : arrL)

{

System.out.println(x);

}

Same as for loop to loop through all elements of arraylist or linkedlist.

## LinkedList

* LinkedList<Object> List = new LinkedList<>();
* List.add(element)
* List.add(index,element) -> can only be used to add to first, last or to existing indexes. Makes list bigger.
* List.remove(index)
* List.addFirst() & List.removeFirst()
* List.addLast() & List.removeLast()
* List.set(index,element) when you want to change existing element at a particular index.
* Linked lists consist of nodes which contain an element and the address of the next node in the linked list.
* The last node contains the memory address null.

## ArrayList vs LinkedList

* Linkedlist good if constantly adding and deleting elements.
* Linkedlist uses two pointers while arraylist uses indexes.
* ArrayList good if frequently searching for elements.
* ArrayList uses less memory because in each arraylist index there is simply an object at that index while in linkedlist there is the data and memory address of next node.
* Linkedlist slower than array when attempting to find an element as the linked list has to iterate through all the elements whereas an array can just arrive at the specified index.
* However, insertion and deletion are much faster in linked list.

### List Iterators

* Iterators in Java work for arraylists and linkedlists.
* They have to be initialised to the same data type as the arraylist or the linkedlist and are specific to a list.
* E.g

// Create a LinkedList

LinkedList<String> linkedlist = new LinkedList<String>();

// Obtaining Iterator

ListIterator it = linkedlist.listIterator();

* Commands are :

-linkedlist.hasNext()

-linkedlist.next()

-linkedlist.hasPrevious()

-linkedlist.previous()

## HashSet

* Difference between list and set:
  + A HashSet is a List with no duplicate members.
  + Because a HashSet is constrained to contain only unique entries, the internal structure is optimised for searching (compared with a list) - it is considerably faster
  + Adding to a HashSet returns a boolean - false if addition fails due to already existing in Set
* You can iterate using the HashSet using Iterator, not ListIterator.
* Iterator does not have hasPrevious() and previous() functions.
* Hashset<String> hashset = new Hashset<String>();

## HashMaps

* In a hashmap, there is no order to elements inside it as opposed to a list. If you were to print out all the elements of a list, they would be in order, however when you iterate over a HashMap, you may get a different order every time you iterate over it.
* HashMap<Object,Object> hashmap = new Hashmap<>();
* You can add elements with their keys using .put().
  + E.g. hashmap.put(“Kishore”, 19);

hashmap.put(“David”, 20);

hashmap.put(“John”,21);

* You can remove an element of a hashmap(key+value) using the .remove() function.

-E.g. hashmap.remove(“Kishore”).

* You can also check whether a hashmap contains a key or a value using the functions .containsKey() or .containsValue()
  + E.g. hashmap.containsKey(“Kishore”) 🡪 true

hashmap.containsValue(22) 🡪 false

* Can get size using .size( ) and can replace an element using .replace()
  + hashmap.replace(“Kishore”, 20);
* Can get all keys using .keySet() and all values using .values() .
* Hashmap doesn’t allow duplicate keys but it does allow duplicate values.
* You can have only one key as “null” in hashmap, but you can have as many null values as needed.

## Classes & Objects

* In Vscode:
  + You can create a new object class within the same Java file defining it as “class Object.”

E.g class Student

* + Or you can have the object class in a separate Java file within the same source folder defining it as “public class Object”.
* When defining and finding the properties of an object you can use the “.” or a get & set method.
  + E.g Student.name = “Mark”;

Student1.setName(“Mark”);

println(Student1.name);

println(Student1.getName( ));

* The get & set methods must be defined within the object class.

## Class Constructor

* When we don’t define a constructor for an object, Java will use its default constructor.
* The constructor of an object has the exact same name as the object, and it will be within the object class.
* A constructor will never return a value.
* You can have multiple constructors with the same name but are made unique depending on what parameters they take in.

## Method Overloading

* Method overloading refers to two fucntions having the same name but having different parameters.
* This is allowed as Java will be able to deduce which method is used based on which parameters you pass in.

