CS 251 FINAL EXAM REVIEW

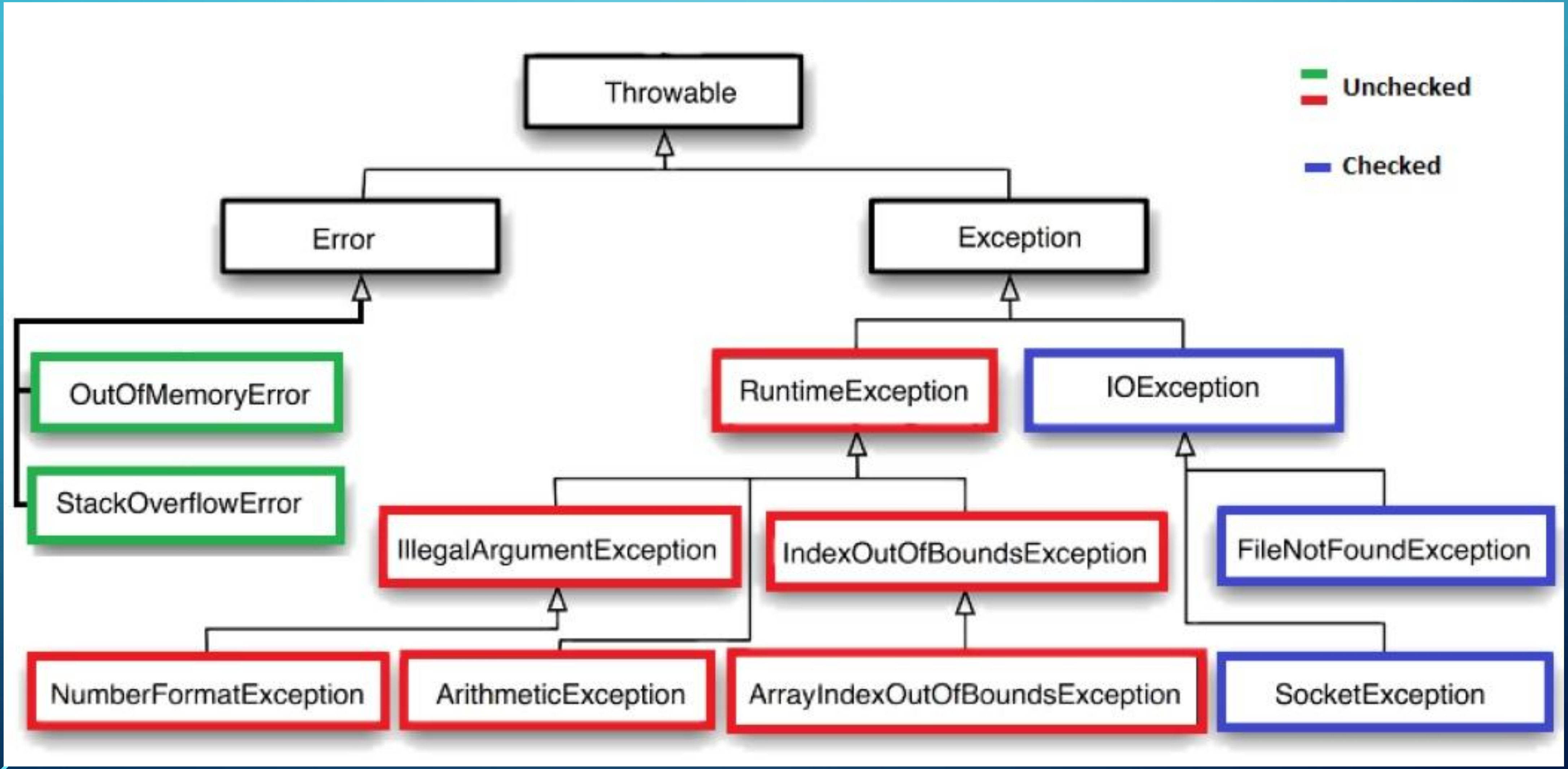
UWM 2021

BY YAZAN SALEM

SOME TIPS FOR THE FINAL EXAM

- Think about what went wrong from the midterm.
- In case of connection problems, be prepared to keep working even if you have internet issues. Copy paste your code somewhere else like notepad, if canvas has issues.
- Think about previous issues you personally had with the midterm and try to already be prepared with strategies for them.
- Fall in love with the java doc.
- Have all relevant java docs already open for you before the exam.
- When studying, check your labs and quizzes too.

Exception Hierarchy – Checked vs Unchecked



CHECKED VS UNCHECKED EXCEPTIONS

- **Checked** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.
- **Unchecked** are the exceptions that are not checked at compiled time. In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.
- => Eclipse

CATCH NOW VS THROWS FOR LATER

- What is the difference between the two ways of handling checked exceptions?
- Try/catch catches the exception in place and deals with it in the catch block
- Throws does not catch the exception. It tells the compiler "it is ok to run, I am already aware of this, but the exception will be handled somewhere else". That is why we annotate in place, so that whoever uses that piece of code anywhere else will know what to expect and potentially catch it there, if he/she wants to or again redirect it with a throws + annotation.

FILE IO

- IO means Input/output
- File is a java class that allows us to create file objects and read/write into and out of them.
- Streams are objects that we use as an intermediate step between files and their source or destination. It transports data from one end to the other.

FILE IO

- Important things that we do with Files:
  - Open
  - Read
  - Write
  - Close
- Their most important and basic methods are the ones that allow us to do these things
- => Eclipse

FILE IO

- When should we close a file and why should we do it?
  - An open file is vulnerable to unwanted modifications of its content and to being damaged. Remember that an open file has a pointer to it and that is dangerous.
  - If we opened a file for writing, we must close it before opening it for reading.
  - We must always close a file after we are done with it.

FILE IO SOME IMPORTANT METHODS

- The constructors:
  - new File(“filename.txt”)
  - new Scanner(file)
  - new PrintWriter(file)
  - new FileWriter(file)
- Scanner- hasNext()
- Scanner- nextLine()
- print()
- println()
- close()
- flush()

BIG O NOTATION

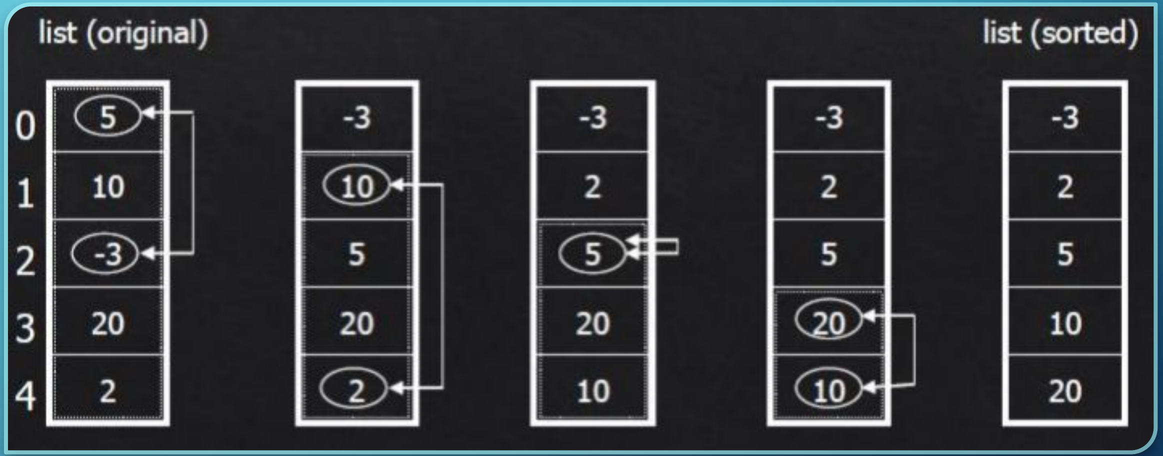
- Algorithms that are dependant on something variable (input size) could have a variable run time.
- Run time: the length of time that takes for an algorithm to finish its process.
- We measure efficiency by comparing the run time of algorithms that are designed for the same purpose.
- We use O notation to represent this measurement, and the worst-case scenario is assumed for each algorithm.
- This is the notation: O(# value with respect to n) where n represents the input size.
- For example: if your input is an array of 10 elements, then your input size is 10, hence n = 10.
- If the variability of the size of the array linearly affects the run time of your algorithm, then its O notation is O(n), if this variability does not affect the run time at all, meaning the run time is always about the same, then we say the algorithm runs in constant time with respect to the input size so its notation will be O(1)
- Ultimately the O notation for an algorithm will be determined by its highest O with respect to its n.
- O(1) < O(logx n) < O(n) < O(n^2) < O(n^x) < O(2^n) < O(x^n) < O(n!) < O(n^n)
- One last thing on this: Nested code gets their O multiplied; line by line code gets their O added.
- => Eclipse

SORTING ALGORITHMS THAT WE SAW

Selection sort	=>	O(n^2)
Insertion Sort	=>	O(n^2)
Quick sort	=>	O(n log n)

SELECTION SORT

```
for i = 0 to length(A)
    min = i
    for j =  i + 1 to length(A)
        if A[j] < A[min]
            min = j
    end for
    swap A[i] and A[min]
end for
```



INSERTION SORT

```
for i = 1 to length(A)
    j = i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j = j - 1
    end while
end for
```

- If selection sort and insertion sort have both O(n^2), which one is better and why?
- Best Case Scenario for Insertion Sort?
- Worst Case Scenario for Insertion Sort?

INSERTION SORT

- If selection sort and insertion sort have both O(n^2), which one is better and why?
  - Insertion sort is more efficient. If the list is partly or completely sorted, it will do less looping.
- Best Case Scenario for Insertion Sort?
  - Array is already sorted. This will run in O(n) now. Selection Sort no matter what will run in O(n2).
- Worst Case Scenario for Insertion Sort?
  - Array is sorted but backwards. Insertion Sort is O(n2) now.

QUICK SORT

- Uses recursion and divide-and-conquer algorithm.
- Because the choosing of the pivot has a big impact on the run time outcome, the average run time of Quick sort is O(n log n), and if the pivot is bad, it can be O(n^2)
- => Eclipse

SHALLOW COPY VS DEEP COPY

- The goal of a deep copy is to have the original and its copy completely independent from each other.
- This means that although their values are the same, their reference addresses are different.
- This will go to the deepest level of the object to be copied.
- If the copy does not reach the deepest level, then is a shallow copy.
- => Eclipse

THE TRADE-OFF OF COPYING

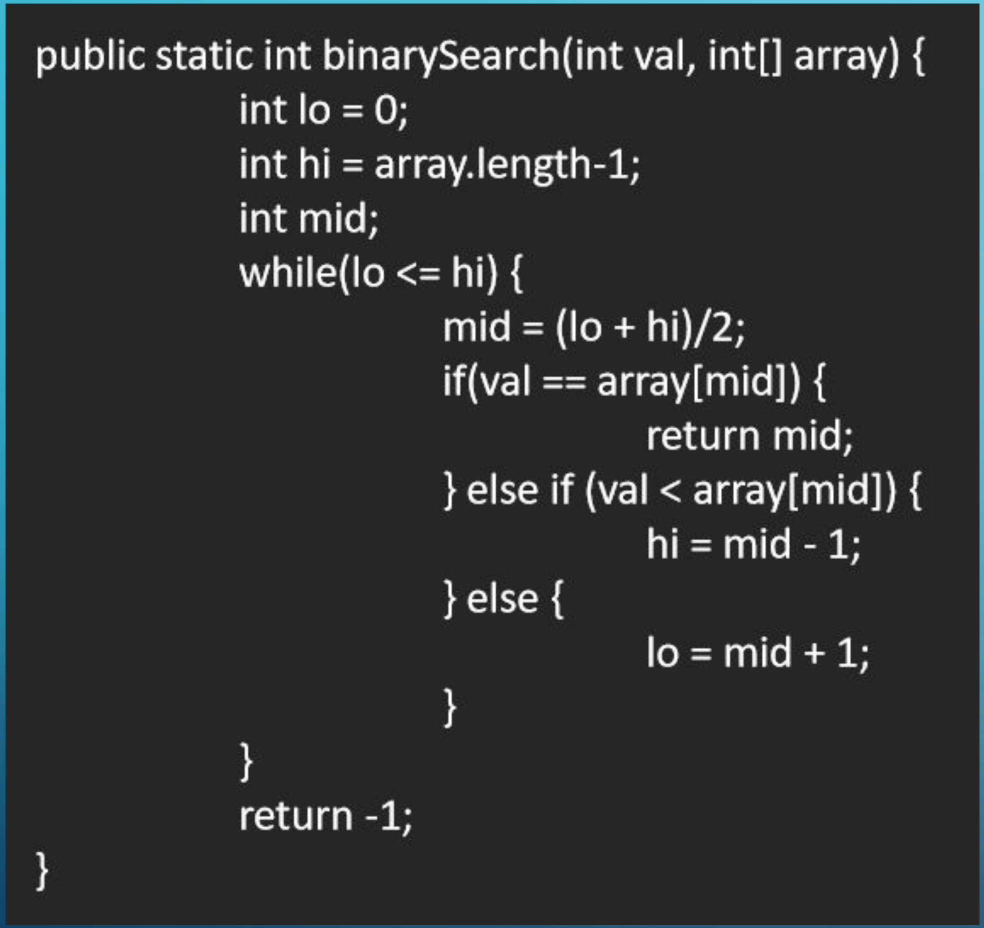
- The deeper the copy, the longer the run time for the process of doing so.
- Run time efficiency:
    - reference copy > shallow copy > deep copy

BINARY SEARCH

- Assumes data is already sorted. Also uses divide-and-conquer algorithm.
- Chooses the middle index and uses it to divide the array in two
- Checks if the element at middle index is the one we want to find.
- If yes, return; else
- Recursively calls itself on the left subarray and on the right subarray
- If no elements to search, return
- This is basically the same as the quick sort algorithm but better, because we do not need a partition method anymore. We can confidently just choose the pivot to be the middle index, because it assumes the data is already sorted.
- Run time is O(log n)

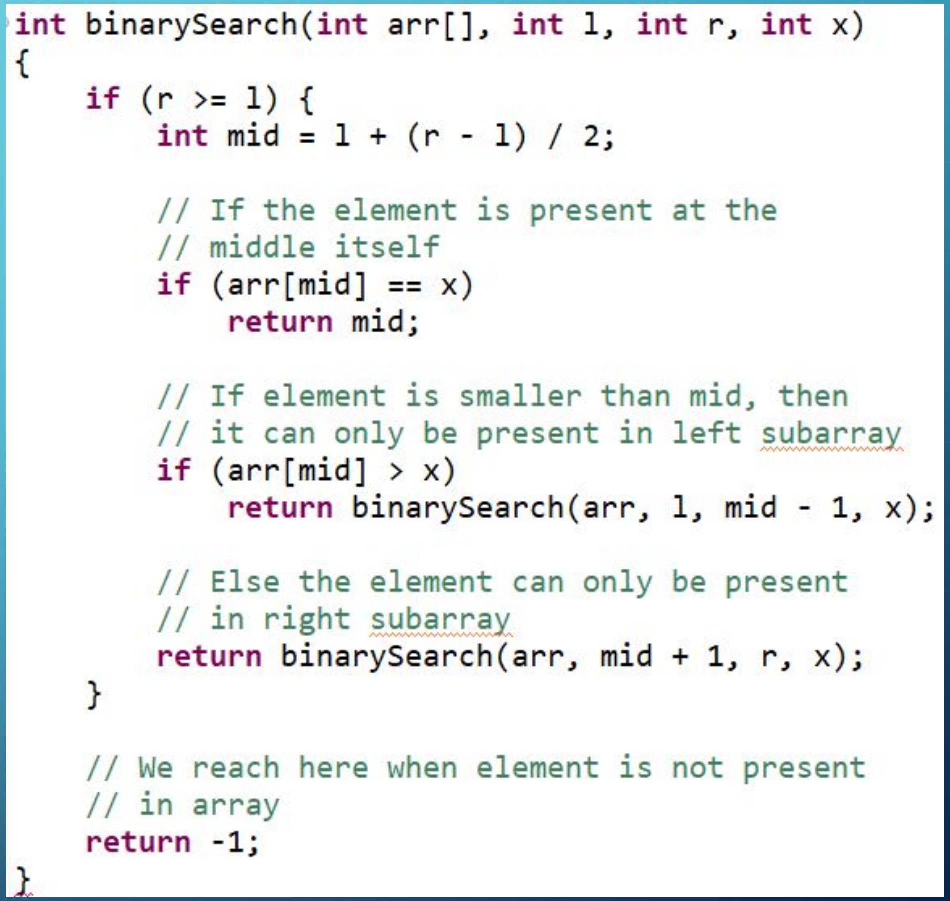
BINARY SEARCH

Non-recursive version



```
public static int binarySearch(int val, int[] array) {
        int lo = 0;
        int hi = array.length-1;
        int mid;
        while(lo <= hi) {
                mid = (lo + hi)/2;
                if(val == array[mid]) {
                        return mid;
                } else if (val < array[mid]) {
                        hi = mid - 1;
                } else {
                        lo = mid + 1;
                }
        }
        return -1;
}
```

Recursive version

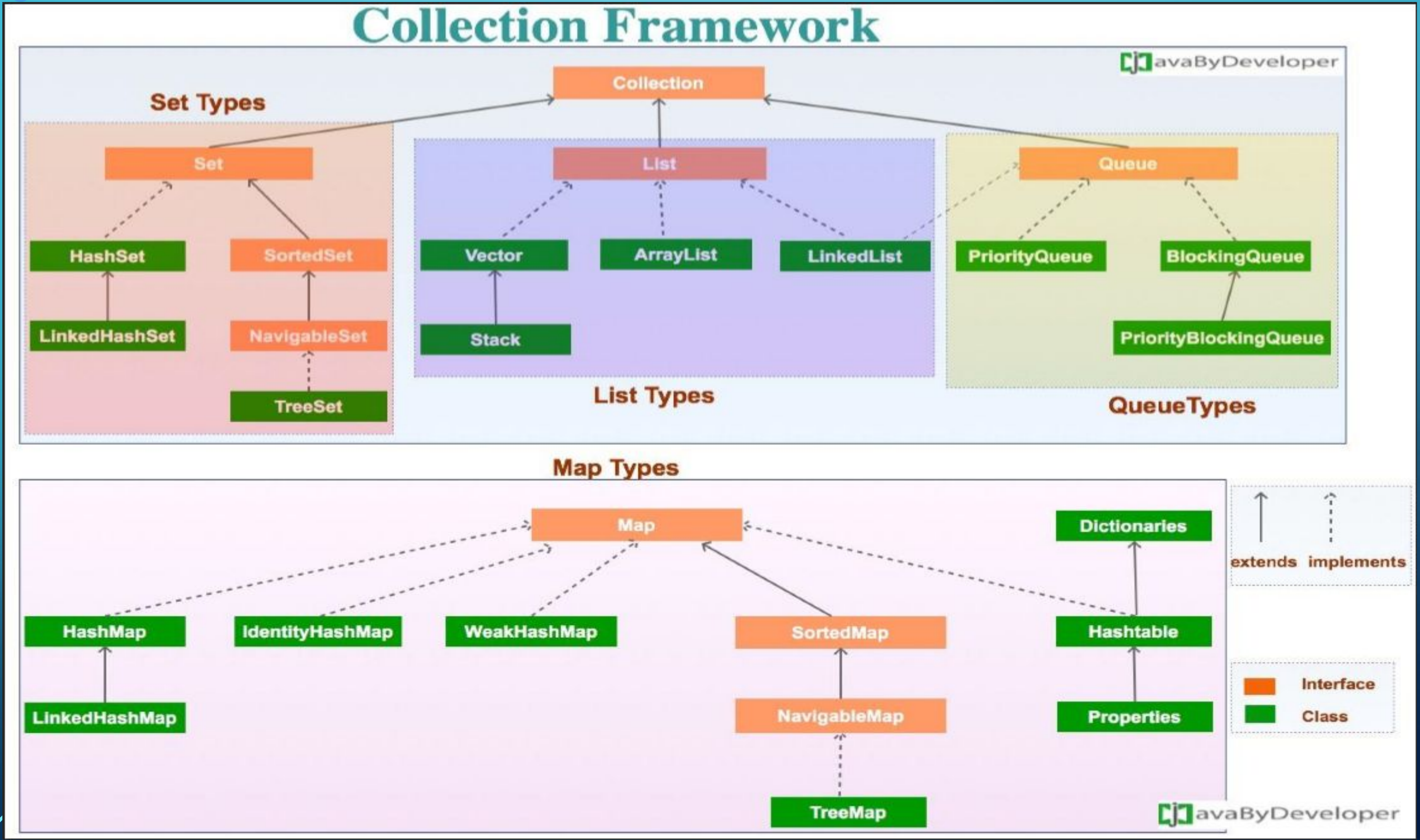


```
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the
        // middle itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not present
    // in array
    return -1;
}
```

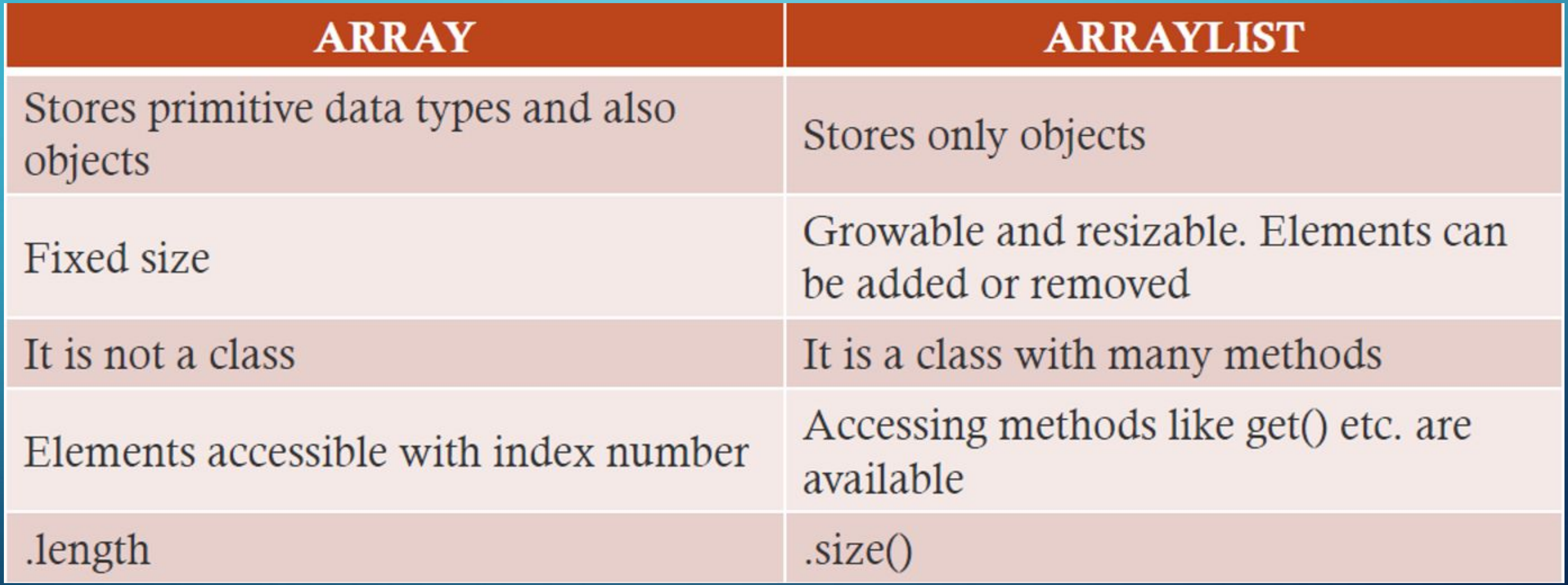
RECURSION

- A method calls itself until it hits the base case.
- The base case is very important; without it, there could be an infinite loop.
- Sometimes recursion is the best approach, sometimes it is not.
- You will have to choose wisely based on efficiency and readability.
- => Eclipse

COLLECTIONS



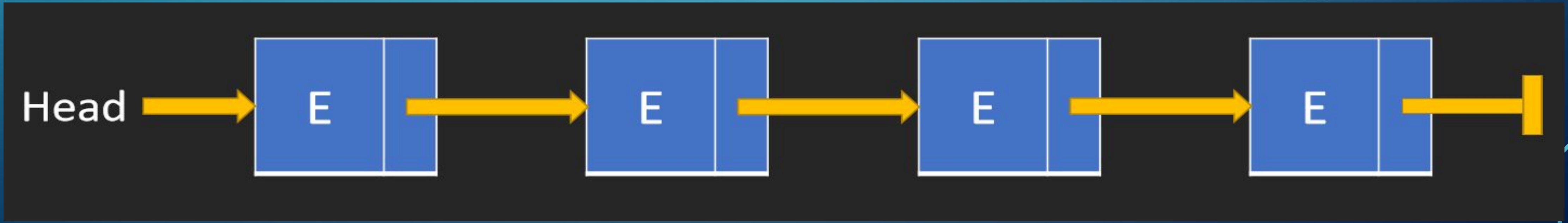
COLLECTIONS - ARRAYLIST



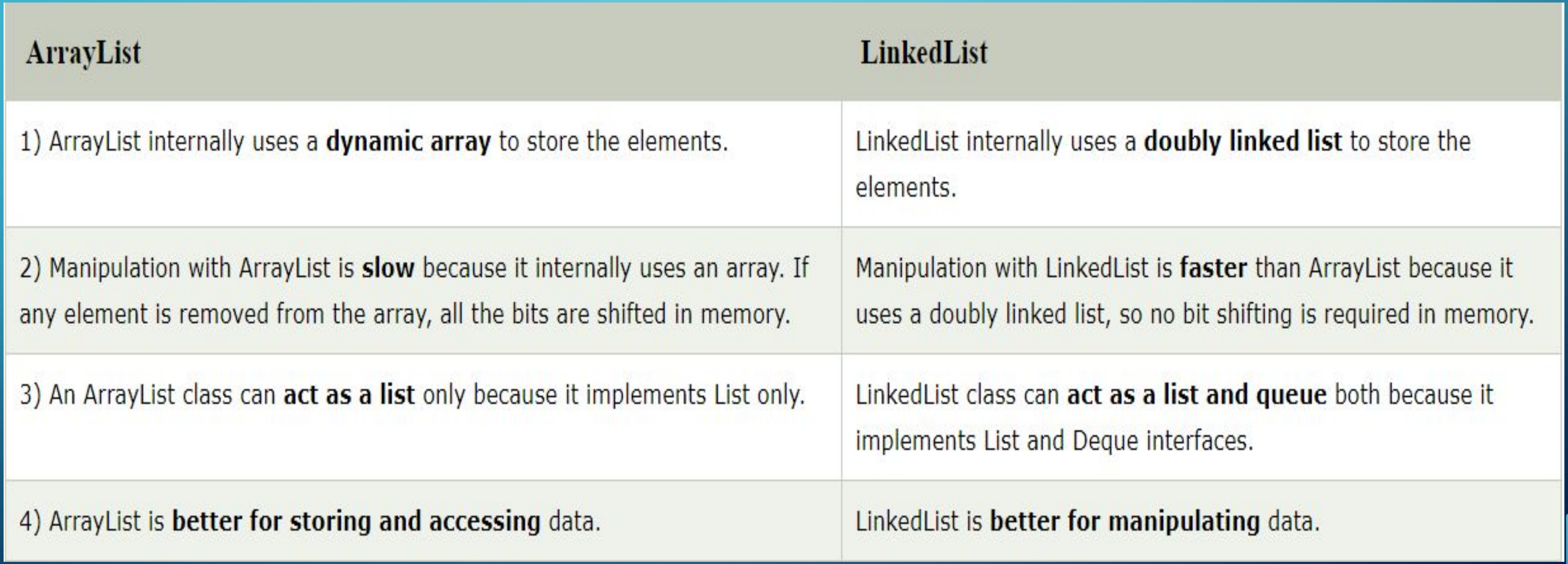
ARRAY	ARRAYLIST
Stores primitive data types and also objects	Stores only objects
Fixed size	Growable and resizable. Elements can be added or removed
It is not a class	It is a class with many methods
Elements accessible with index number	Accessing methods like get() etc. are available
.length	.size()

COLLECTIONS - LINKEDLIST

- ArrayLists use arrays to store all data.
- Linked Lists use nodes, where each node points to the next node.
- Below is a singly linked list. There are many variations on this.



ARRAYLISTS VS LINKEDLISTS



ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

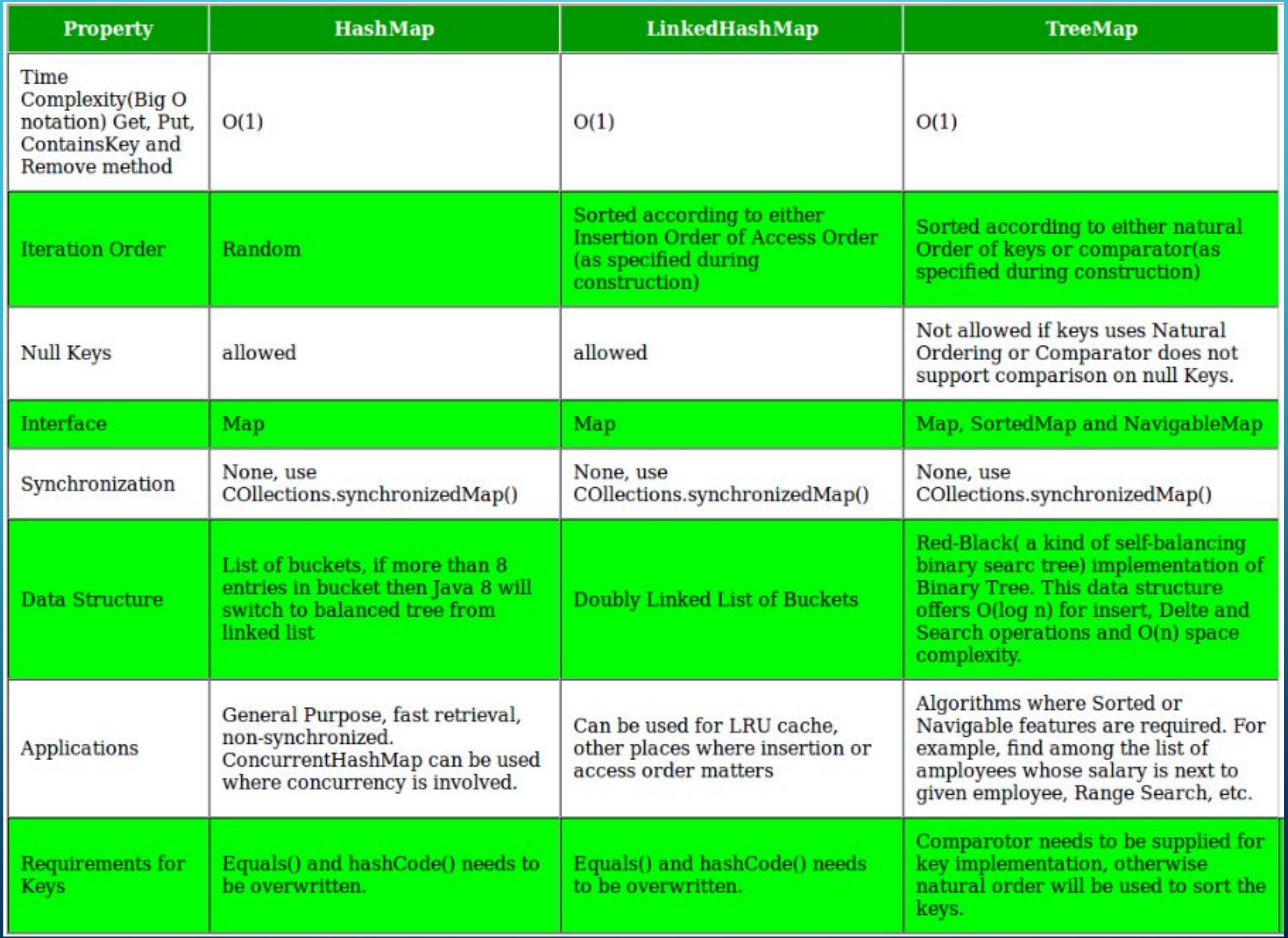
BASIC LIST METHODS

- size
- contains
- isEmpty
- add
- remove
- clear
- get
- set
- subArray

MAPS

- A collection object that maps keys to values.
- Values can be duplicates.
- Keys must be unique.
- One key maps to one value
- A map stores objects of the type <Key,Value> pairs which are Map.Entry objects.
- We can access the value by using the key.

HASHMAP VS LINKEDHASHMAP VS TREEMAP



Property	HashMap	LinkedHashMap	TreeMap
Time Complexity(Big O notation) Get, Put, ContainsKey and Remove method	O(1)	O(1)	O(1)
Iteration Order	Random	Sorted according to either Insertion Order of Access Order (as specified during construction)	Sorted according to either natural Order of keys or comparator(as specified during construction)
Null Keys	allowed	allowed	Not allowed if keys uses Natural Ordering or Comparator does not support comparison on null Keys.
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()	None, use COllections.synchronizedMap()
Data Structure	List of buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black(a kind of self-balancing binary searc tree) implementation of Binary Tree. This data structure offers O(log n) for insert, Delte and Search operations and O(n) space complexity.
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of amployees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten.	Equals() and hashCode() needs to be overwritten.	Comparotor needs to be supplied for key implementation, otherwise natural order will be used to sort the keys.

BASIC MAP METHODS

- get
- put
- remove
- containsKey
- containsValue
- keyset
- => Eclipse

GUI

- Graphic User Interface
- => Eclipse

IMPORTANT QUESTIONS

- What are divide and conquer algorithms?
- Is recursion better than non-recursion?
- Why do Generic classes exist?
- How do Generic classes relate to the principles of OOP?
- When should we use a map and when should we not?
- How to choose the right List type?
- Which List type has access to Queue methods?
- Understand the collection methods, what parameters they take, what they return, what exceptions they throw.

THANK YOU

Good Luck!