COLLECTION FRAMEWORK

What is the need for collections?

I have 1000 variable to declare. How should I do that?

```
int var1=10;

int var2=20;

int var3=30;

......

.....

int var1000=40;
```

Using Arrays

```
int[] myArray = new int[1000];
```

Limitations with Array

- Fixed in Size
- Homogenous data type
- Arrays not implemented based on some standard data structure. There is no ready methods.

COLLECTIONS

- Growable in nature. Can increase or decrease the size.
- Can hold different data types.
- Standard data structure. There are ready methods to use.

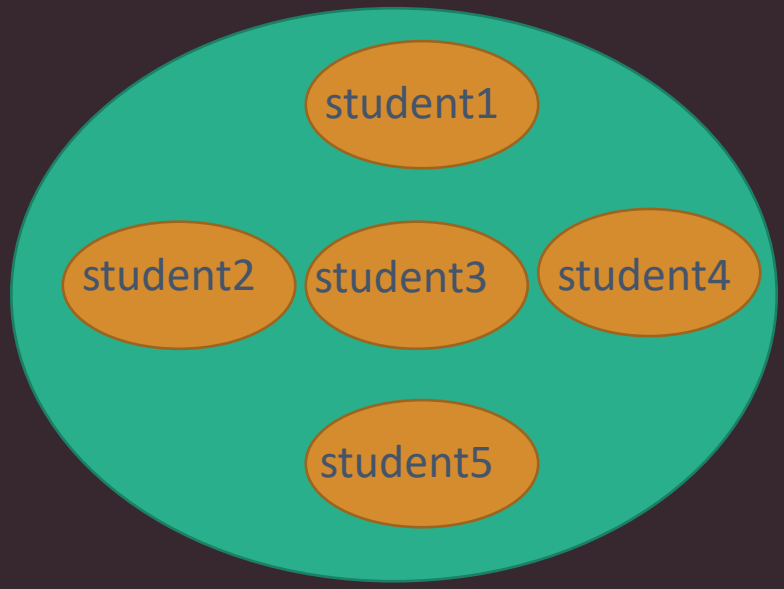
Which concept is recommended to use?

- If you know the size in advanced, better to use arrays.
- Consider performance issue while using the collections.

	ARRAYS	COLLECTIONS
Size	Fixed in Size	Growable in Size
Point of Memory		
Point of Performance		
Data Type	Homogenous	Homogenous & Heterogenous
Data Structure		
Can hold:	Primitive & Object Types	Only Object Types

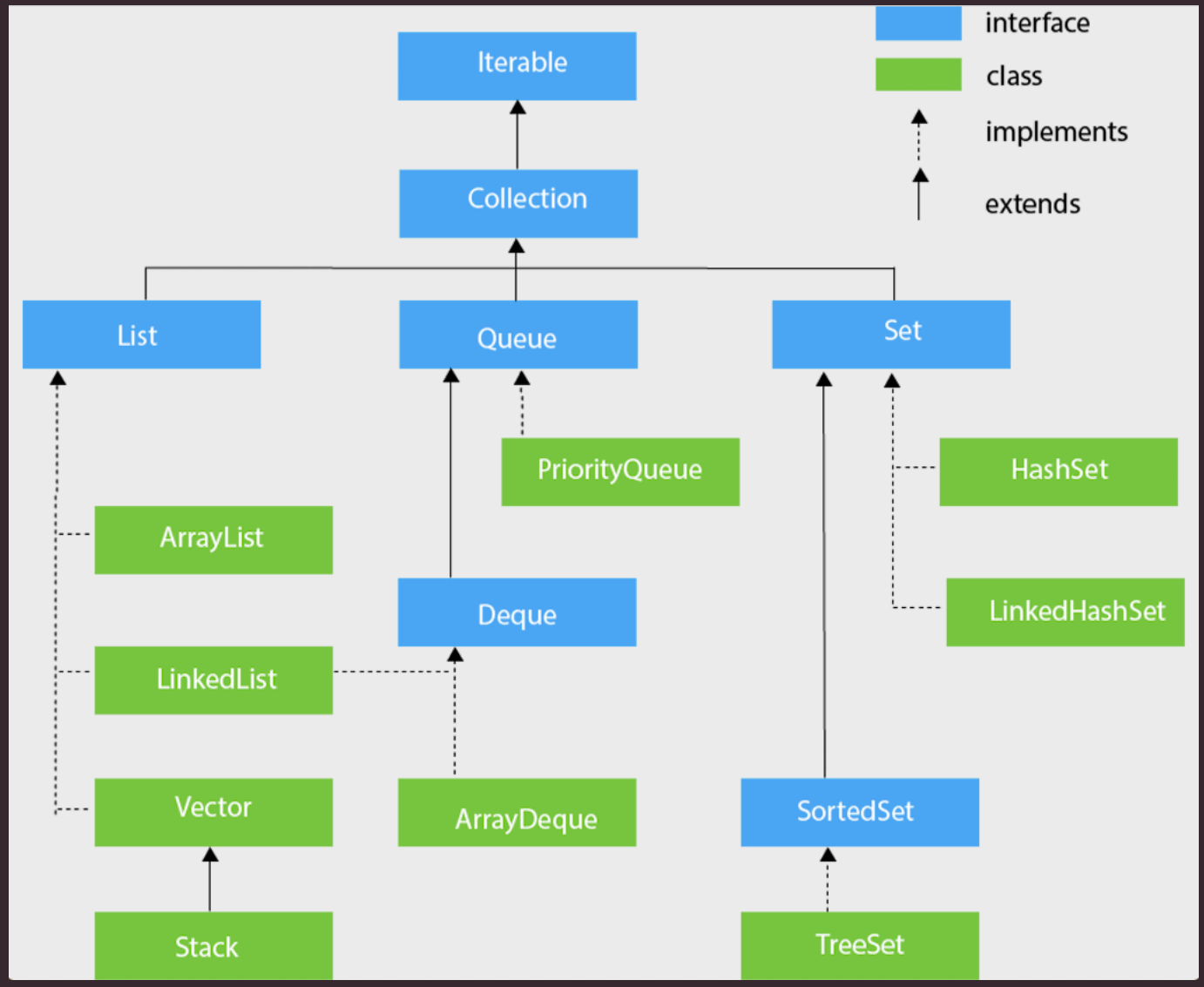
What is collection?

- Collection is a group of individual objects as a single entity.



What is collection framework?

- It defines several classes and interfaces which can be used to represent a group of objects as single entity.



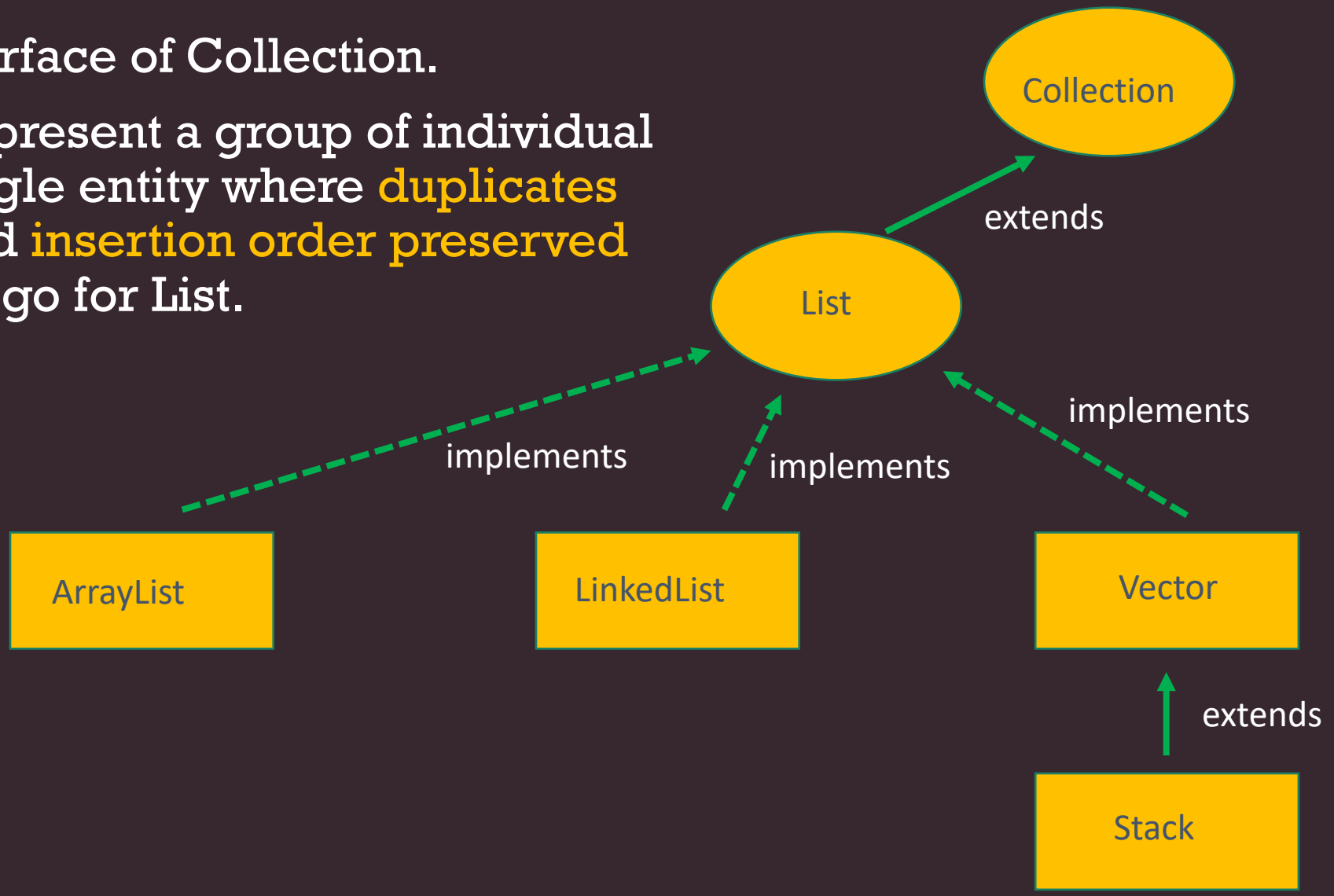
Collection

- Collection interface defines the most common methods which are applicable for any collection object
- In general collection interface is considered as root interface of collection framework.
- There is no concrete class which implements collection interface directly.

Method	Definition
size()	Get the number of elements in the Collection
isEmpty()	True if size()==0, false otherwise
add(element)	Add the element at the beginning of this collection
addAll(collection)	Add all the elements of the argument collection to this collection
remove(element)	Remove the element from this collection
removeAll(collection)	Remove all the elements of this collection
contains(element)	True if the element is in this collection, false otherwise
containsAll(collection)	True if all the elements of the argument collection are in this collection
clear()	Remove all elements from this collection

LIST

- List is child interface of Collection.
- If we want to represent a group of individual objects as a single entity where duplicates are allowed, and insertion order preserved then we should go for List.

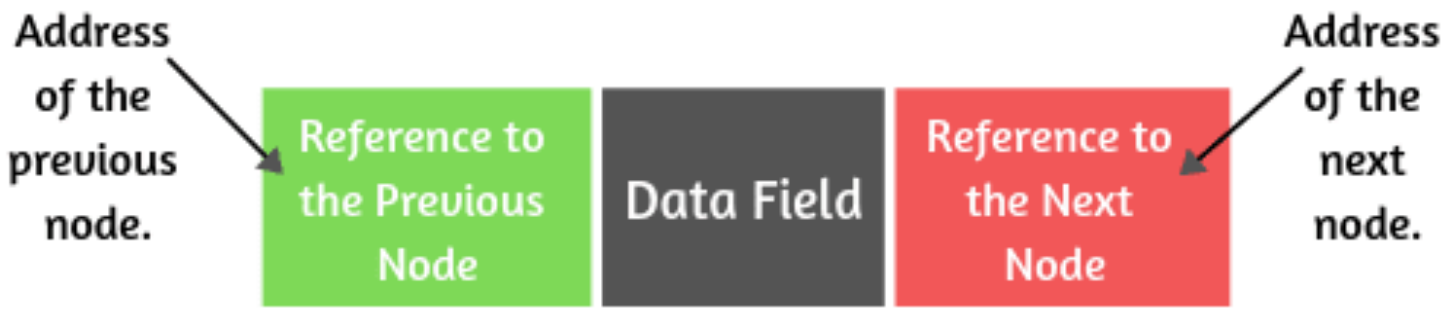


ArrayList

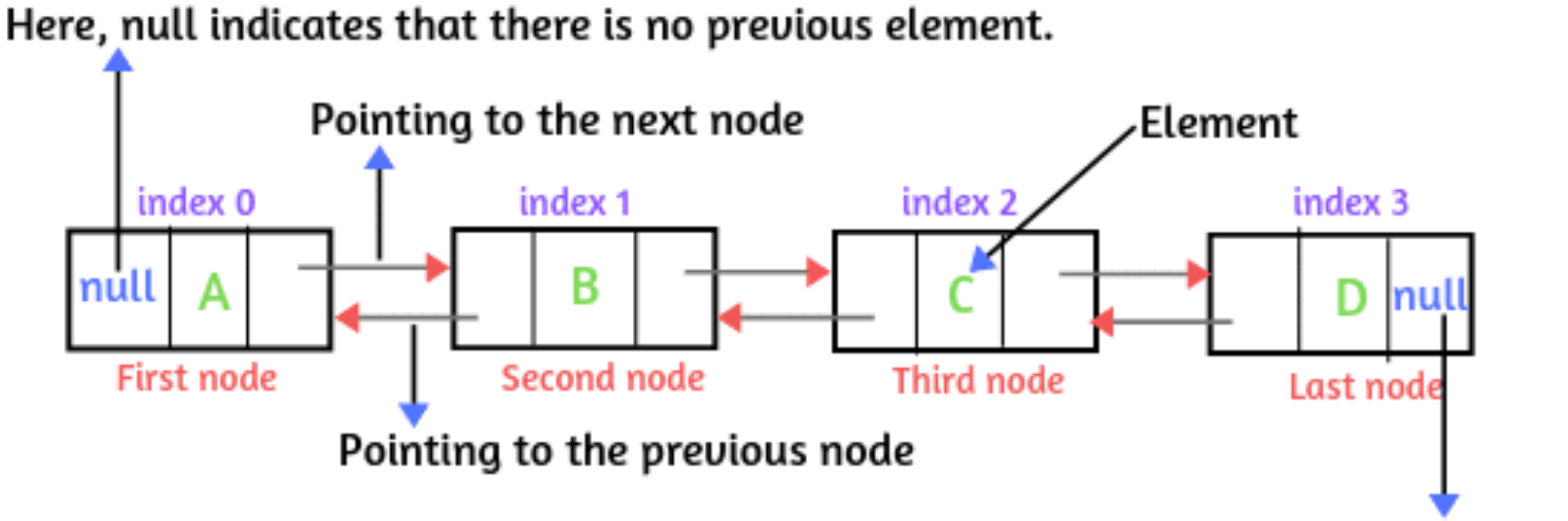
- An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed.
- ArrayList internally uses an array to store the elements. Just like arrays, It allows you to retrieve the elements by their index.
- Java ArrayList allows duplicate values.
- Java ArrayList is an ordered collection. It maintains the insertion order of the elements.
- You cannot create an ArrayList of primitive types like int, char etc. You need to use boxed types like Integer, Character, Boolean etc.

LinkedList

- Implementing class of a List interface
- Stores data in individual nodes that are connected to each other
- Each node will have a pointer to the next node/value
- Consists of values connected to each other using pointers
- Implements List and Deque interfaces



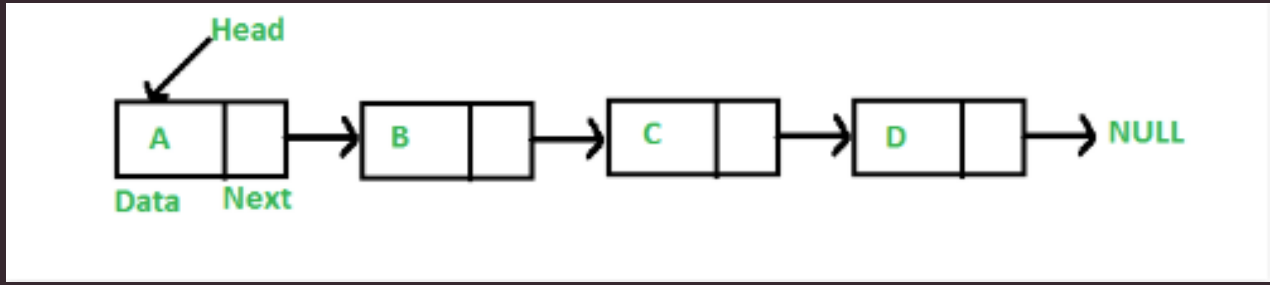
Representation of Java LinkedList Node



A array representation of linear Doubly LinkedList in Java

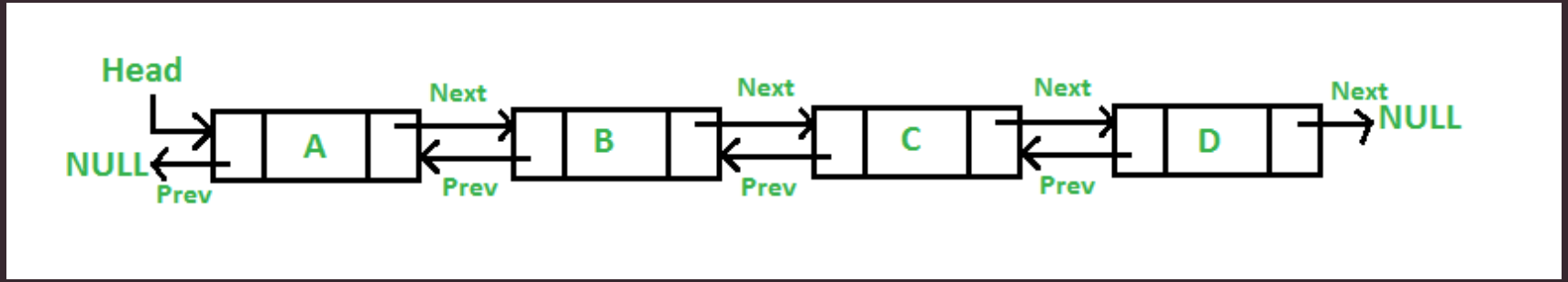
Singly Linked list and Doubly Linked List

- Singly Linked List: A singly linked list is a set of nodes where each node has two fields ‘data’ and ‘link’. The ‘data’ field stores actual piece of information and ‘link’ field is used to point to next node.



Singly Linked list and Doubly Linked List

- Doubly Linked List: (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

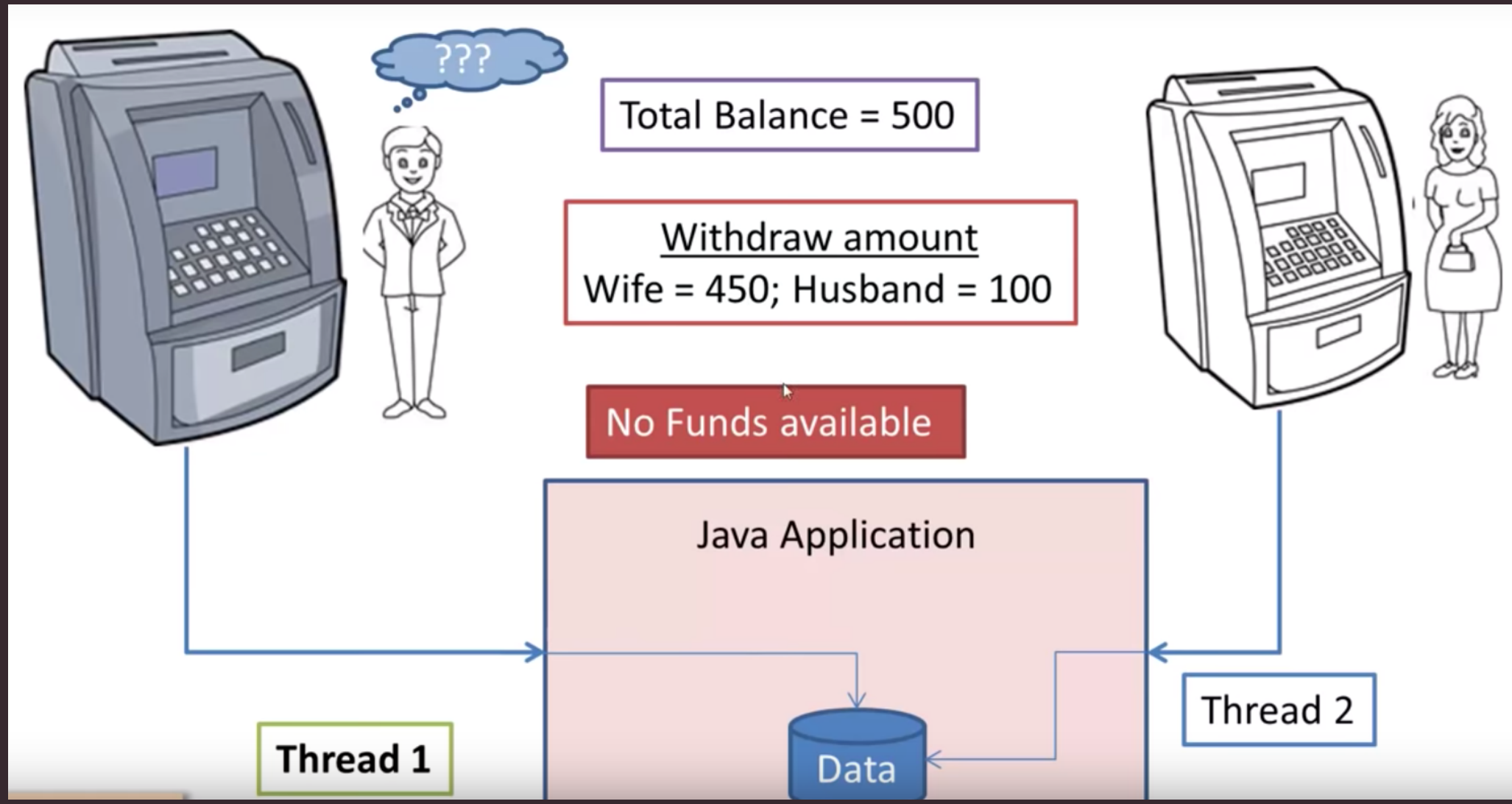


LinkedList vs ArrayList

ArrayList	LinkedList
ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
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.
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.
ArrayList is **better for storing and accessing** data. **get()**	LinkedList is **better for manipulating** data. **add(), remove()**

Vector

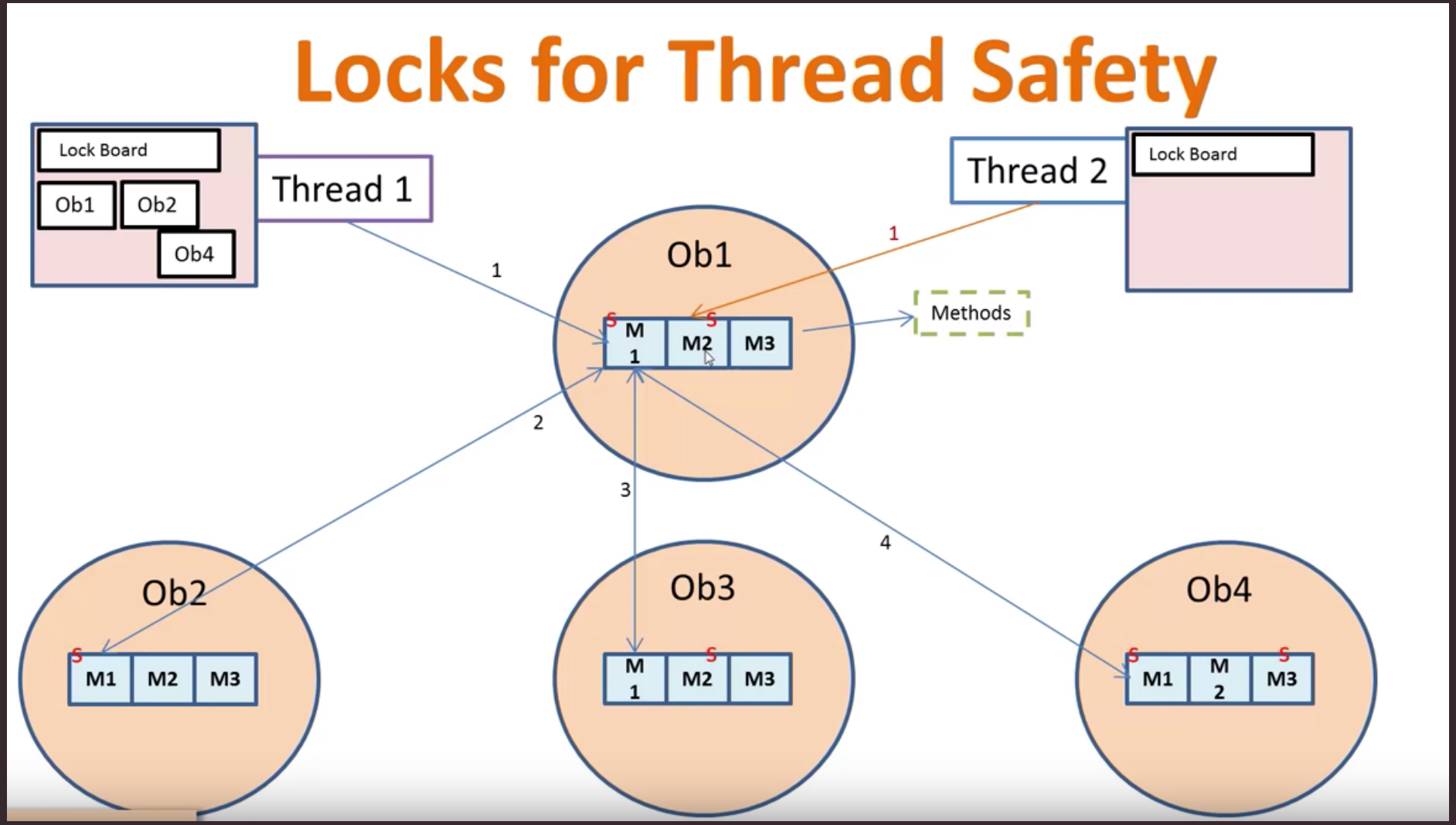
- Vector implements a dynamic array. It is similar to ArrayList, but with two differences :
  - Vector is synchronized(thread-safe)
  - Vector contains many legacy methods that are not part of the collection framework.


???
Total Balance = 500
Withdraw amount
Wife = 450; Husband = 100
No Funds available
Java Application
Thread 1
Thread 2
Data

Synchronize

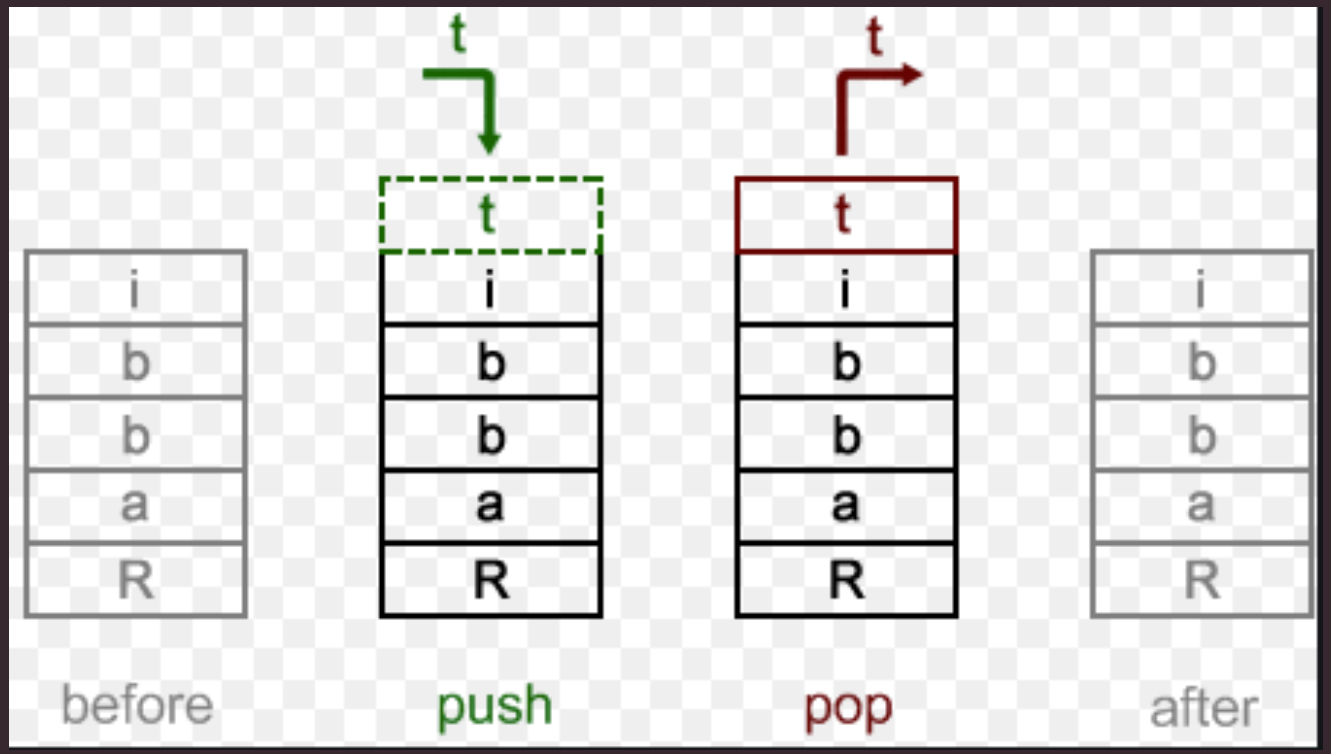
- Code Synchronization helps in preventing multiple threads executing a code simultaneously.
- Code Synchronization is implemented with help of Locks.
- A thread that is trying to access the code that is marked as Synchronized should acquire the lock from object.
- Locks:
  - Every object has a lock. Only one lock per object.
- Synchronize can only be applied for methods and block of code.

Locks for Thread Safety



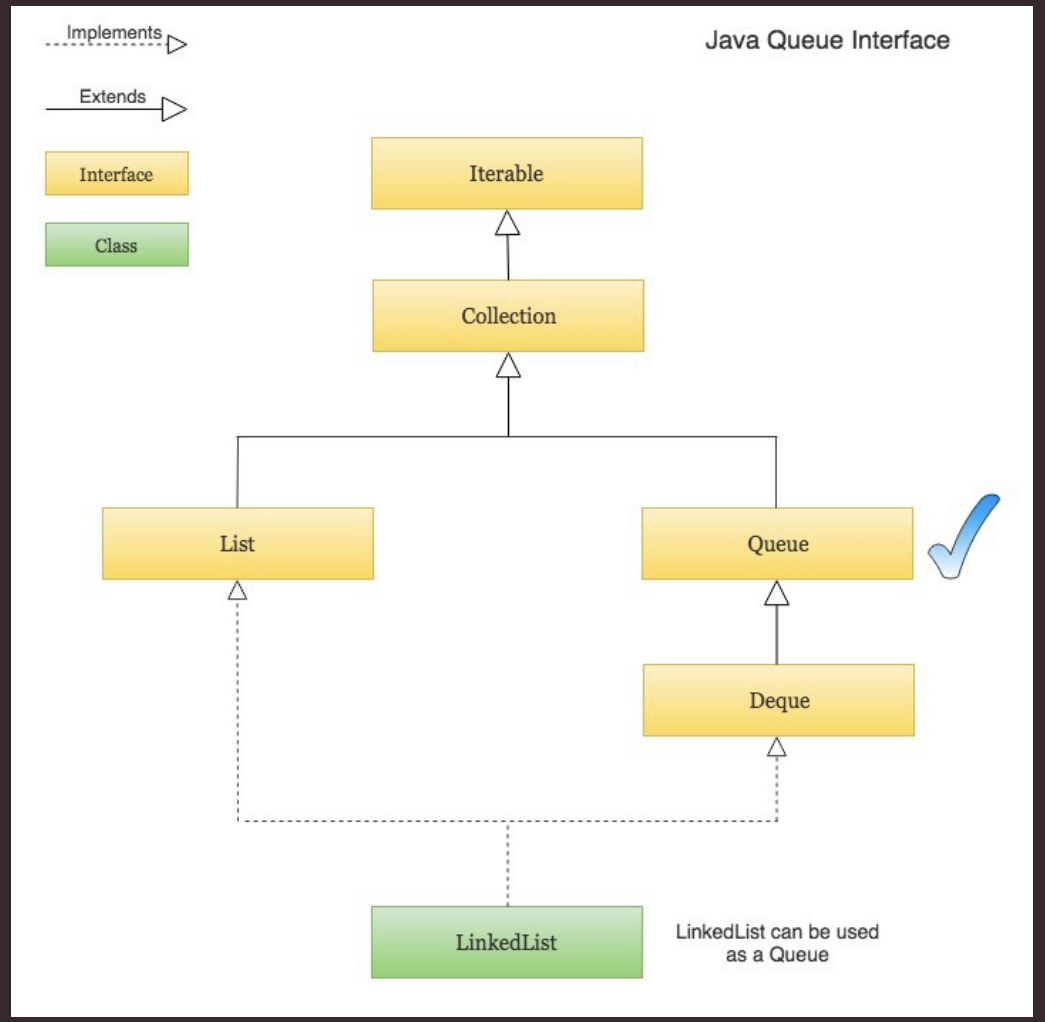
Stack

- Stack is a subclass of Vector that implements a standard last-in, first out. (LIFO)



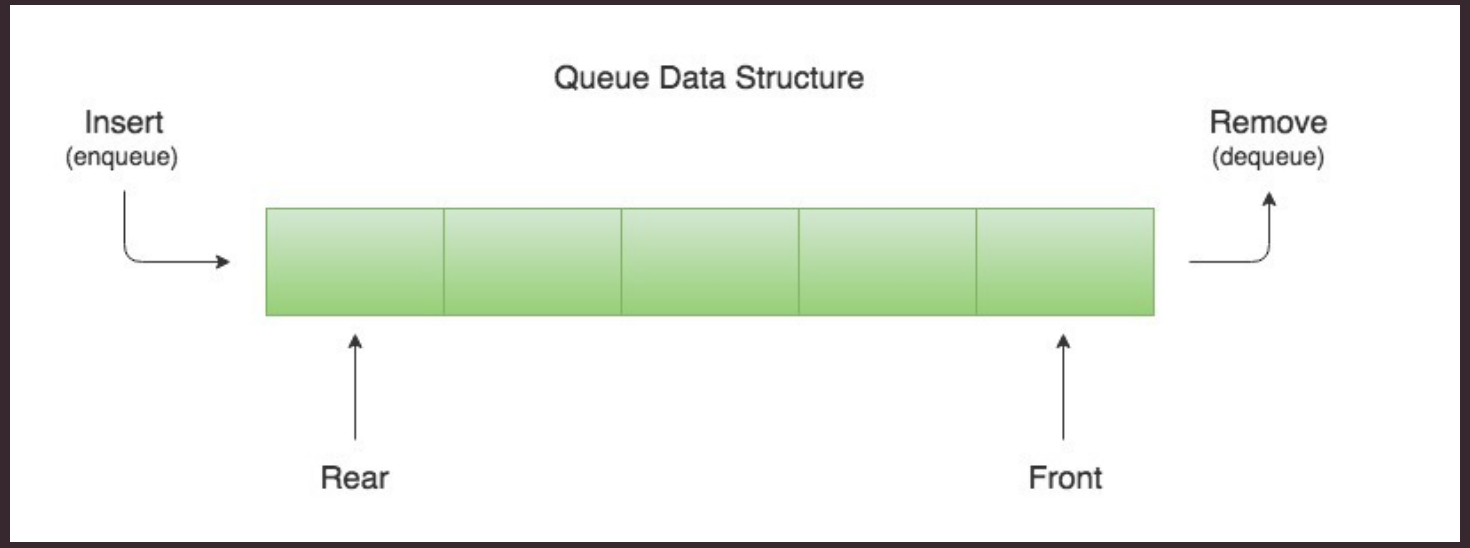
Queue

- It is child interface of Collection.
- If we want to represent a group of individual objects prior to processing then we should go for Queue.



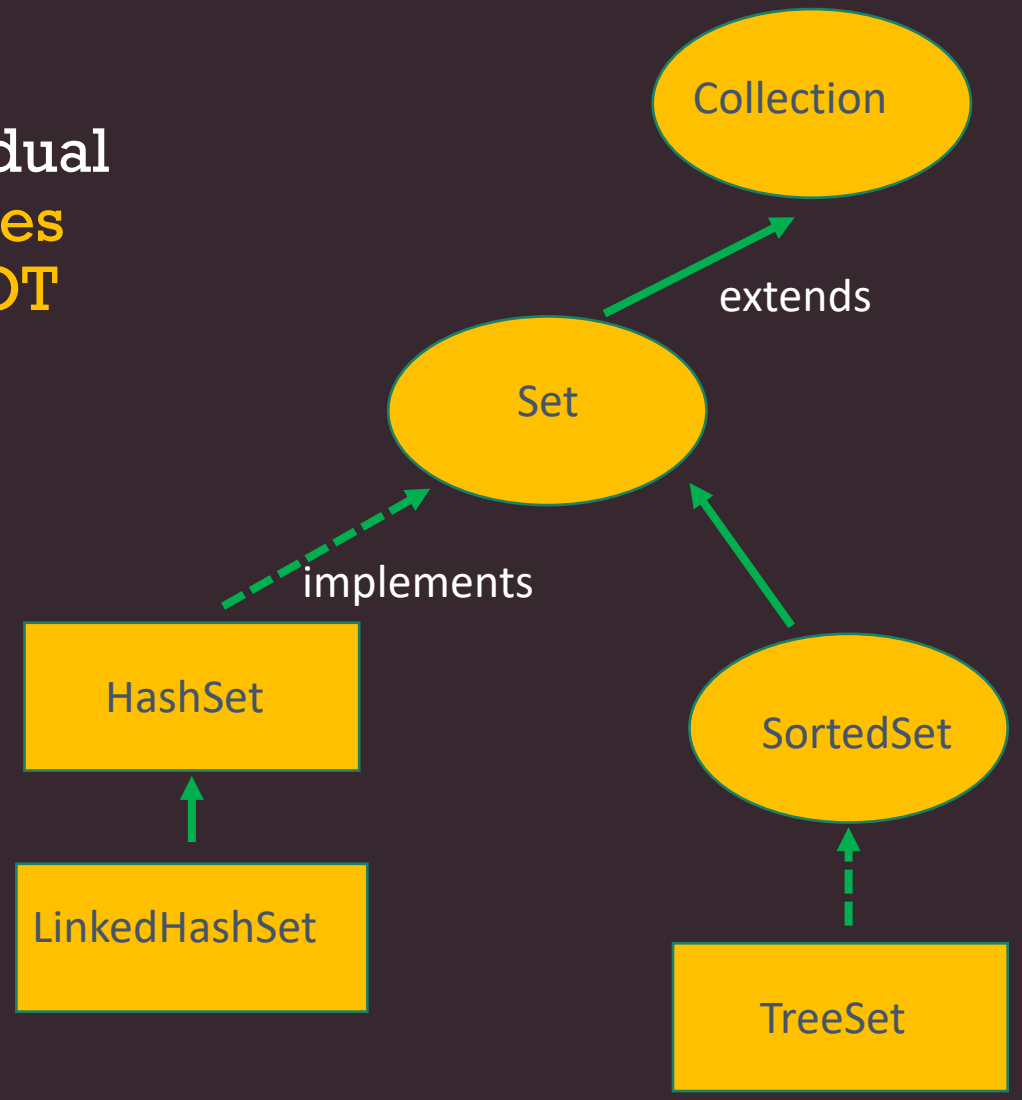
Queue

- A Queue is a First In First Out (FIFO) data structure.



SET

- Set is child interface of Collection.
- If we want to represent a group of individual objects as a single entity where duplicates are NOT allowed, and insertion order NOT preserved then we should go for Set.



HashSet

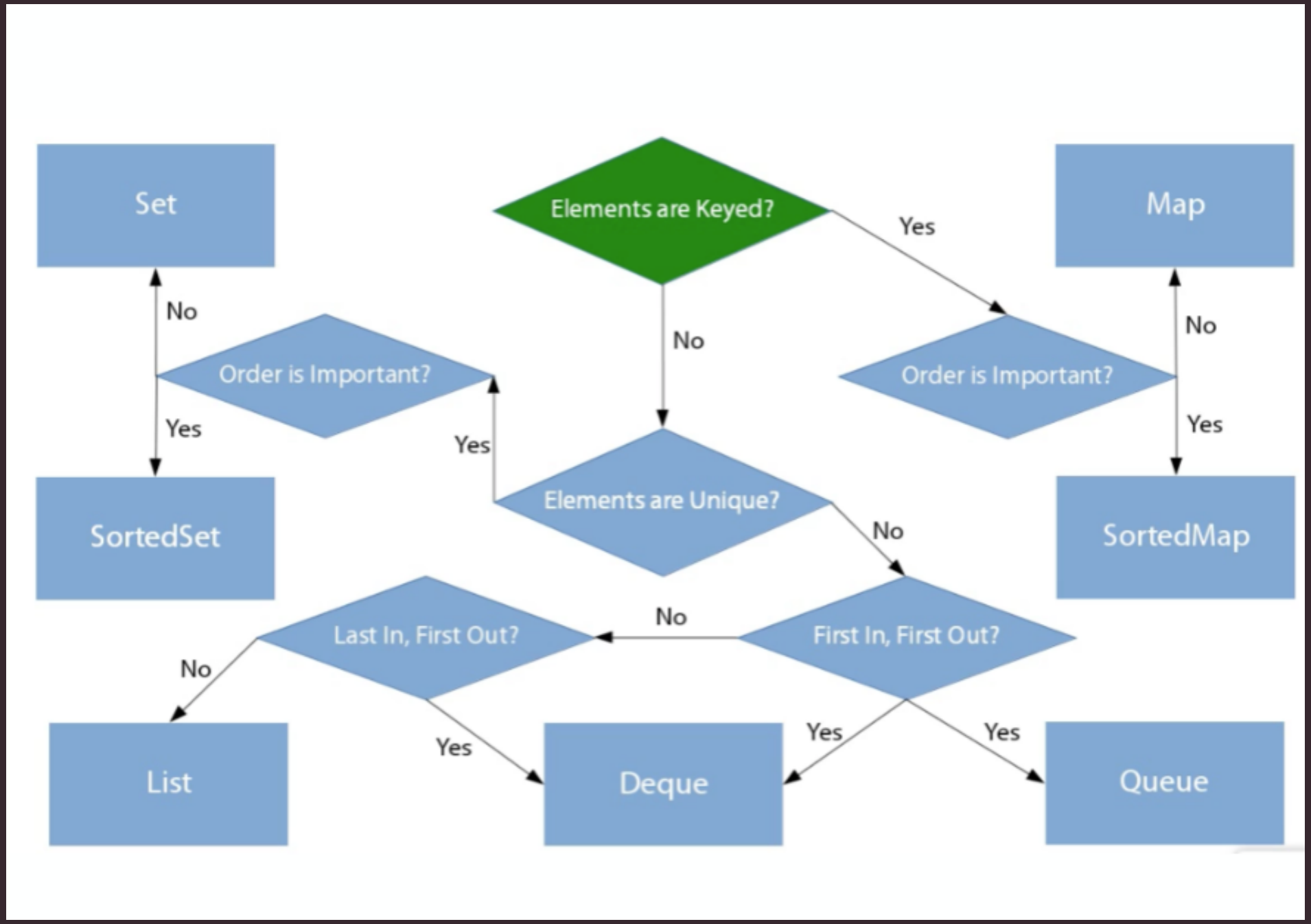
- HashSet class is used to create a collection that uses a hash table for storage.
- HashSet contains unique elements only.
- HashSet class is not synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.

TreeSet

- TreeSet implements the SortedSet interface so duplicate values are not allowed.
- Objects in a TreeSet are stored in a sorted and ascending order.
- TreeSet does not preserve the insertion order of elements but elements are sorted by keys.
- TreeSet serves as an excellent choice for storing large amounts of sorted information which are supposed to be accessed quickly because of its faster access and retrieval time.

DIFFERENCES BETWEEN LIST AND SET

LIST	SET
Duplicates are allowed	Duplicates are not allowed
Insertion order preserved	Insertion order not preserved


Set
Elements are Keyed?
Yes
Map
No
No
Order is Important?
Order is Important?
No
Yes
Yes
Yes
SortedSet
Elements are Unique?
No
SortedMap
No
Last In, First Out?
First In, First Out?
No
Yes
Yes
Yes
List
Deque
Queue

Loop Through Collection

1. For each loop
2. Any other loop(for, while, do while) by using get(index) method
3. Iterator
4. forEach method that came with java 8 (lambda expression)

How do you sort an ArrayList?

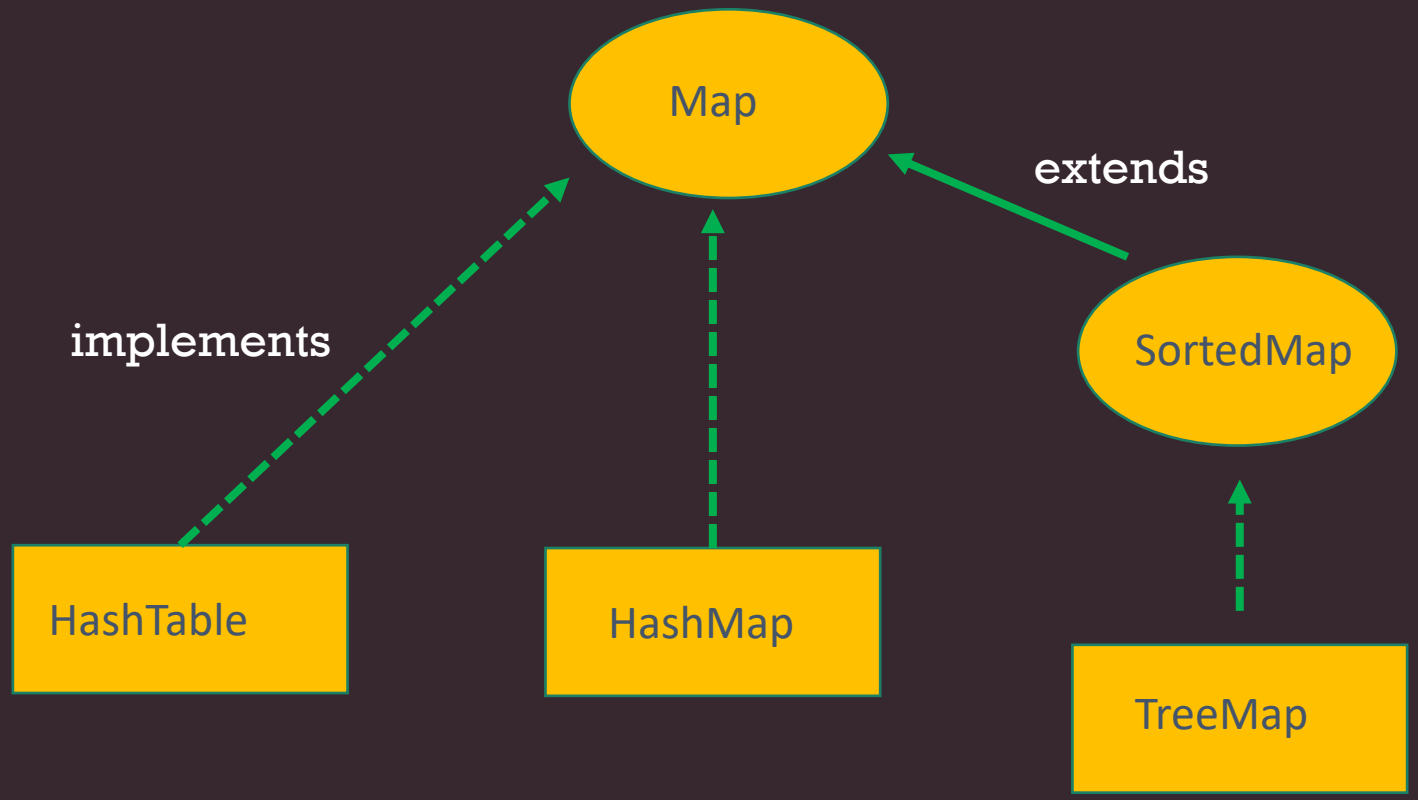
```java
List<Integer> l1 = new ArrayList<>();
l1.add(10);
l1.add(20);
l1.add(50);
l1.add(1);
l1.add(8);

Collections.sort(l1);

System.out.println(l1);  // [1, 8, 10, 20, 50]
```

What is the difference between Iterator and For Each Loop?

- When using iterator object, we can remove values while looping
- When using for each loop, we can not remove values from the collection
- We need to create iterator object to able to use it
- For each loop works with a temporary variable

Collection of Pairs : Map

- Data structure based on key + value pairs
- Map interface does not extend Collection interface



Difference between HashMap and HashTable

HashMap	HashTable
Every method in HashMap are not synchronized	Every method in HashTable are synchronized
HashMap is fast	HashTable is slow
HashMap allows one null key and multiple null values	HashTable does not allow any null key or value
HashMap implements Map interface	HashMap implements Map interface

Difference between HashMap and TreeMap

HashMap	TreeMap
HashMap does not maintain any sorting order	TreeMap elements are sorted according to natural sorting order
Internally it used hash table	Internally is uses Red Black Tree
Contains one null key and many null values	Can not contain null keys but may contains many null values
HashMap implements Map interface	It implements SortedMap interface