

0301502 ADVANCED JAVA

UNIT	MODULES	WEIGHTAGE
1	File Handling	20 %
2	Java Collection Framework	20 %
3	Event Handling, Swing and GUI Components	20 %
4	Swing, GUI Components and Layout Manager	20 %
5	Database Connectivity (JDBC)	20 %

UNIT -2 Java Collection Framework

- Introduction
- Collection Class
- Linked List
- Array List
- Stack
- Queue
- Set
- Maps
- Iterator
- ListIterator

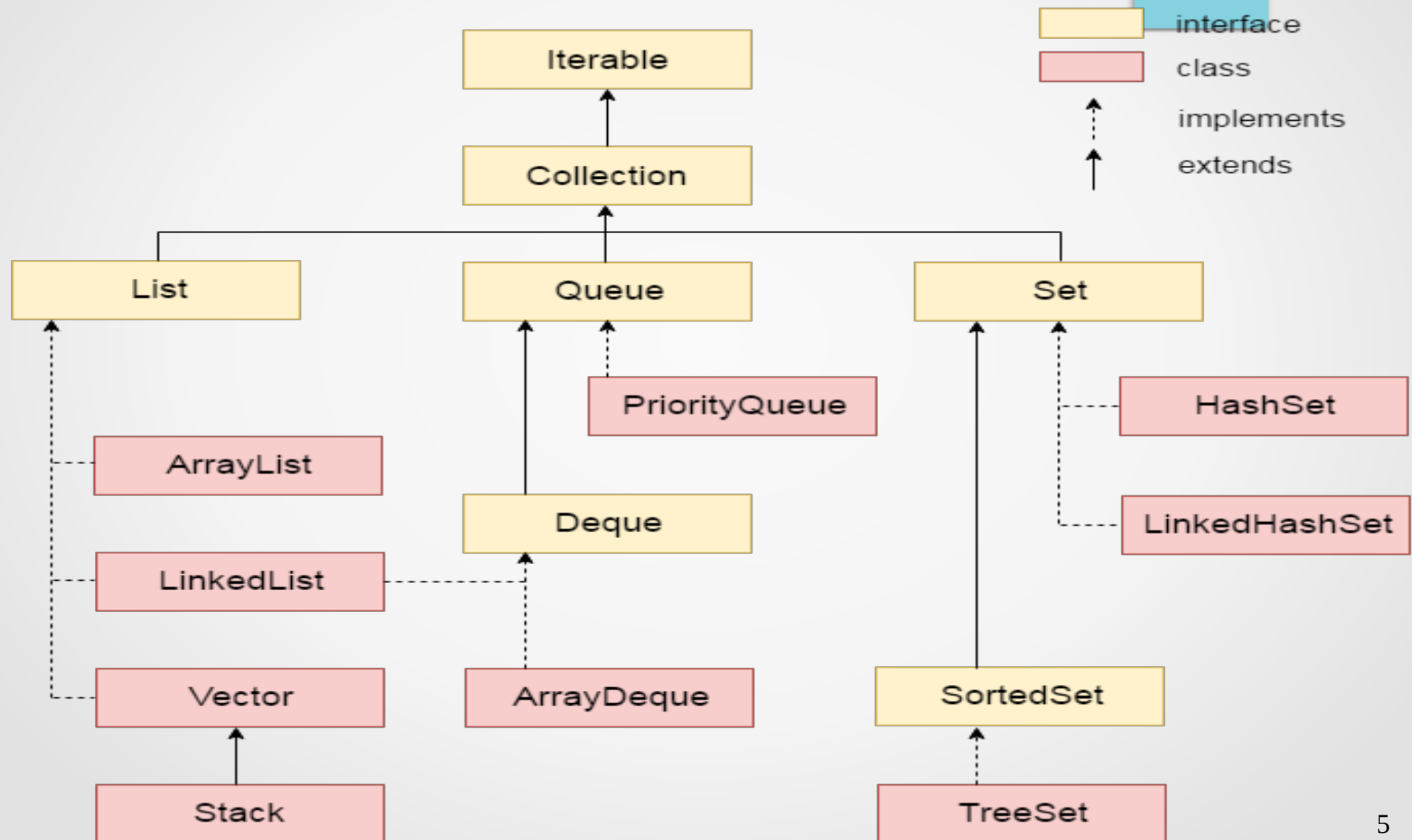
UNIT -2 Introduction

- The collection framework **proide a well- designed set of interface** and classes **for stroing and manipulating of data** as a single unit.
- It provides the folloing abstract data type
 - *Maps*
 - *Sets*
 - *Lists*
 - *Trees*
 - *Arrays*
 - *Hashtables*

UNIT -2 Introduction – Feature of Collections Framework

- Implementation of fundamental collection like dynamic array, linked list, tree etc is **highly efficient with high performance**.
- All **collection** has almost same look and feel and their way of working is similar to each other.
- **Extending a collection is very easy.**
- Whole of the **collections** are designed around a set of standard interfaces.
- Collection framework also **allows creating one's own collection**.

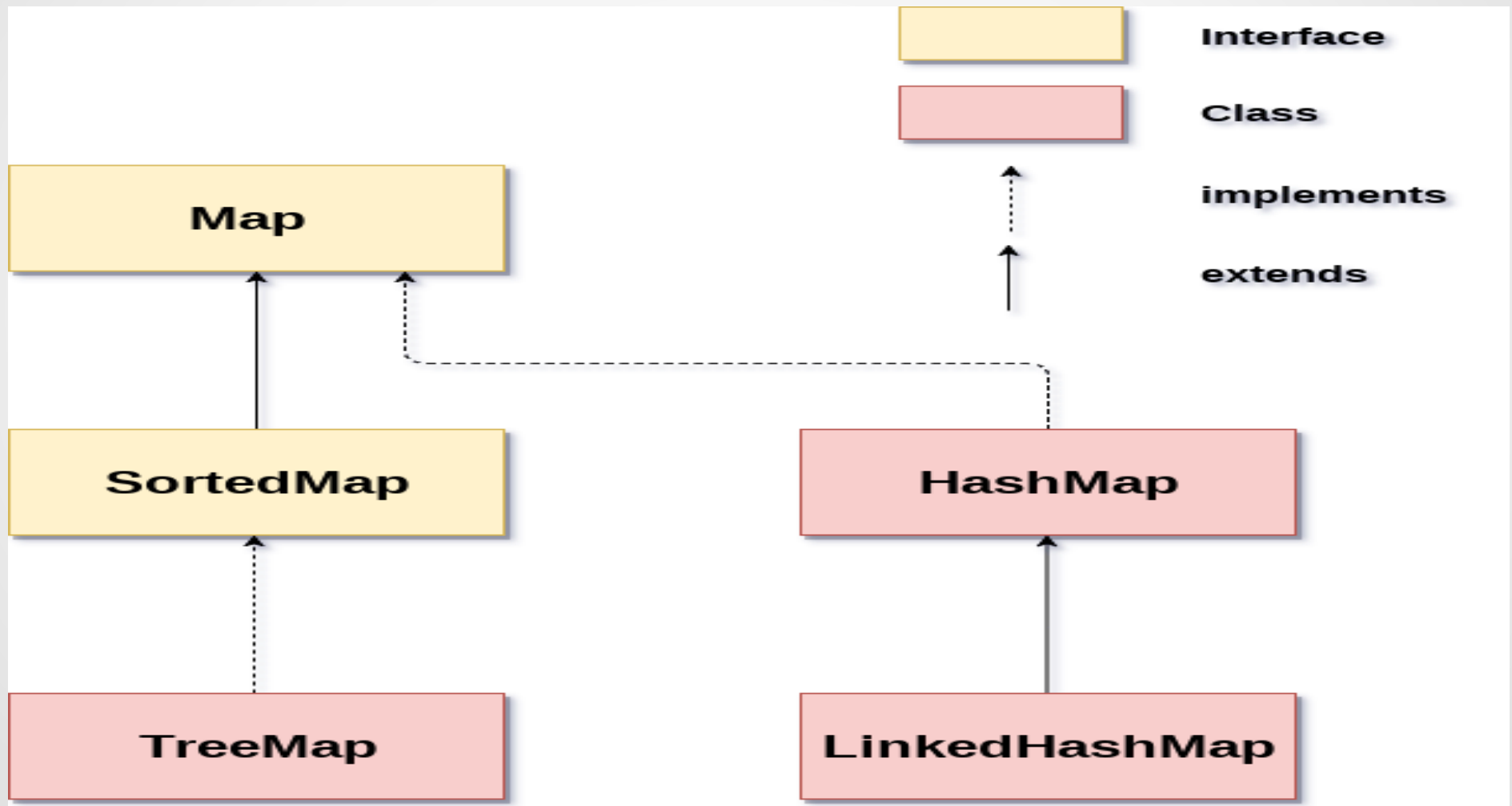
UNIT -2 Introduction - Java Collection Framework



UNIT -2 Introduction – MAP

- *Map* is just **collection of Pairs**.
- The interfaces *Map* and *Collection* **are distinct**

UNIT -2 Java Collection Framework - MAP



UNIT -2 Introduction

- The **Following points need to be remembered** regarding Collection Framework:
 - The *Collection* interface is a group of objects, **with duplicates allowed.**
 - The *Set* interface extends Collection but **forbids duplicates**
 - The *List* interface extends Collection, **allows duplicates and introduces positional indexing.**
 - The *Map* interace **extends neither Set nor Collection.**

UNIT -2 Collection Interface

- The Collection interface is used to represent any group of objects or elements.
- This interface is implemented by all collection classes.
- The interface supports basic operations like adding and removing.

UNIT -2 Collection Interface - Methods

Method	Purpose
<i>boolean add(Object obj)</i>	Add obj to the invoking collection.
<i>Boolean add(Collection c)</i>	Add all the elements of c to the invoking collection.
<i>Void clear()</i>	Removes all elements from the invoking collection.
<i>boolean contains(Object obj)</i>	Returns true if obj is an element of the invoking collection.
<i>boolean containsAll(collection c)</i>	Returns true if the invoking collection contains all elements of c.
<i>boolean equals(Object obj)</i>	Returns true if the invoking collection is equals.
<i>boolean isEmpty()</i>	Returns true if the invoking collection is empty.

UNIT -2 Collection Interface - Methods

Method	Purpose
<i>Iterator iterator()</i>	Returns an iterator for the invoking collection.
<i>boolean remove(Object obj)</i>	Remove one instance of obj from the invoking collection.
<i>boolean removeAll(Collection c)</i>	Remove all elements of c from the invoking collection.
<i>boolean retainAll(Collection c)</i>	Remove all elements from the invoking collection except those in c.
<i>Int size()</i>	Returns the number of elements held in the invoking collection.
<i>Object [] toArray()</i>	Returns an array that contains all the elements stored in the invoking collection.
<i>Object [] toArray(Objectarray [])</i>	Returns an array containing only those collection elements whose type matches that of the array.

UNIT -2 List Interface

- The List interface extends the Collection interface to define an ordered collection.
- Permitting duplicates.
- The interface adds position oriented operations.
- The first element in the list starts at index 0.
- Elements can be added and accessed by their position in this list.

UNIT -2 List Interface - Methods

Method	Purpose
<i>void add(index, object obj)</i>	Insert into the invoking list at the index passed in index.
<i>boolean addAll(int index, Collection c)</i>	Inserts all elements of c into the invoking list at the index passed in index.
<i>object get(int index)</i>	Returns the object stored at the specified index within the invoking collection.
<i>int indexOf(object obj)</i>	Returns the index of the first instance of obj in the invoking list. Return -1 if obj is not an element.
<i>int lastIndexOf(Object obj)</i>	Return the index of the last instance of obj in the invoking list. Return -1 if obj is not an element.
<i>listIterator listIterator()</i>	Return an iterator to the start of the invoking list

UNIT -2 List Interface - Methods

Method	Purpose
<i>listIterator listIterator(int index)</i>	Return an iterator to the invoking list that begins at the specified index.
<i>object remove(int index)</i>	Removes the element at position index from the invoking list and returns the deleted elements.
<i>object set(int index, Object obj)</i>	Assigns obj to the location specified by index within the invoking list
<i>list subList(int start, int end)</i>	Returns a list that includes elements from start and end.

UNIT -2 List Interface

- **List**
 - List is an ordered collection of objects in **which duplicate values can be stored**. Since List preserves the insertion order it allows positional access and insertion of elements.
 - **List Interface is implemented by**
 - **ArrayList**
 - **LinkedList**
 - **Vector**
 - **Stack classes.**

UNIT -2 List Interface

- **List interface has various class has follow constructor:**
 - *List a = new ArrayList();*
 - *List b = new LinkedList();*
 - *List c = new Vector();*
 - *List d = new Stack();*

UNIT -2 List Interface

- Example:
 - ListDemo.java
 - ListDemo2.java

UNIT -2 LinkedList

- **LinkedList**
 - **Linked list is a fundamental data structure** that contains records.
 - **A record contains data as well as a reference to the next record.**
 - **A record can be inserted or removed at any point** in the Linked List.
 - **Random access is not allowed like array. Only sequential access is allowed.**
 - This class act as a stack, queue and double-ended queue.
- **LinkedList class has follow constructor:**
 - `LinkedList()`
 - `LinkedList(Collection c)`

UNIT -2 LinkedList Class - Methods

Method	Purpose
<i>void add(int index, Object element)</i>	It is used to insert the specified element at the specified position index in a list.
<i>void addFirst(Object o)</i>	It is used to insert the given element at the beginning of a list.
<i>void addLast(Object o)</i>	It is used to append the given element to the end of a list.
<i>int size()</i>	It is used to return the number of elements in a list
<i>boolean add(Object o)</i>	It is used to append the specified element to the end of a list.

UNIT -2 LinkedList Class - Methods

Method	Purpose
<i>boolean contains(Object o)</i>	It is used to return true if the list contains a specified element .
<i>boolean remove(Object o)</i>	It is used to remove the first occurrence of the specified element in a list.
<i>Object getFirst()</i>	It is used to return the first element in a list.
<i>Object getLast()</i>	It is used to return the last element in a list.
<i>int indexOf(Object o)</i>	It is used to return the index in a list of the first occurrence of the specified element , or -1 if the list does not contain any element.
<i>int lastIndexOf(Object o)</i>	It is used to return the index in a list of the last occurrence of the specified element , or -1 if the list does not contain any element.

UNIT -2 LinkedList

- Example:
 - DemoLinedlist.java
 - LinkedListExample.java
 - DemoLinkedList_stack.java
 - DemoLinkedList_queue.java

UNIT -2 ArrayList

- **Array List**

- Java ArrayList class uses a **Dynamic Array** for storing the elements.
- **It inherits Abstract List class and implements List interface.**
- Java ArrayList class can **contain duplicate elements**.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows **random access because array works at the index basis**.
- In Java ArrayList class, manipulation is **slow because a lot of shifting needs to be occurred if any element is removed from the array list**.

UNIT -2 ArrayList

- **LinkedList class has follow constructor:**
 - *ArrayList()*
 - *ArrayList(Collection c)*
 - *ArrayList(int capacity)*

UNIT -2 ArrayList Class Methods

Method	Details
<i>void add(int index, Object element)</i>	It is used to insert the specified element at the specified position index in a list.
<i>boolean addAll(Collection c)</i>	It is used to append all of the elements in the specified collection to the end of this list , in the order that they are returned by the specified collection's iterator.
<i>void clear()</i>	It is used to remove all of the elements from this list.
<i>int lastIndexOf(Object o)</i>	It is used to return the index in this list of the last occurrence of the specified element , or -1 if the list does not contain this element.
<i>Object[] toArray()</i>	It is used to return an array containing all of the elements in this list in the correct order.
<i>Object[] toArray(Object[] a)</i>	It is used to return an array containing all of the elements in this list in the correct order.

UNIT -2 ArrayList Class Methods

Method	Details
<i>boolean add(Object o)</i>	It is used to append the specified element to the end of a list.
<i>boolean addAll(int index, Collection c)</i>	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
<i>Object clone()</i>	It is used to return a shallow copy of an ArrayList.
<i>int indexOf(Object o)</i>	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
<i>void trimToSize()</i>	It is used to trim the capacity of this ArrayList instance to be the list's current size.

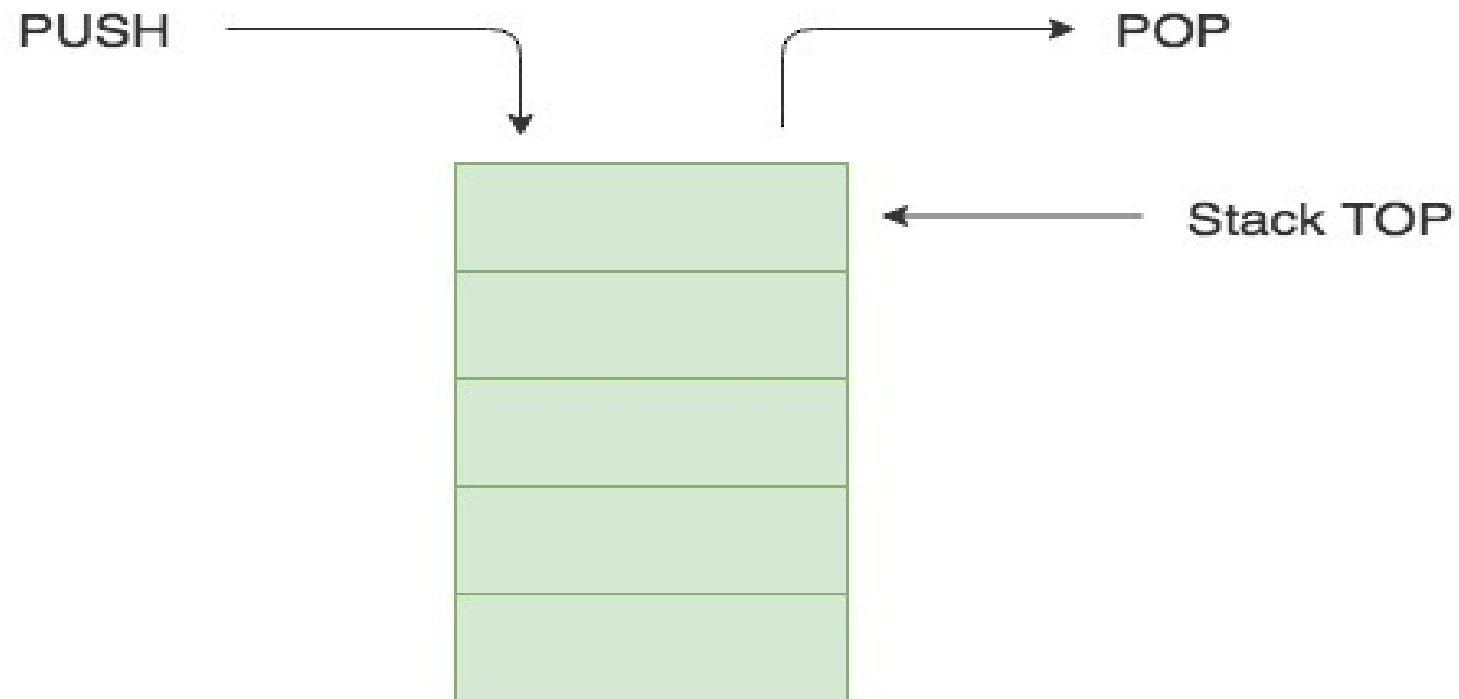
UNIT – 2 ArrayList

- Example:
 - TestCollection3.java

UNIT -2 Stack

- **Stack**
 - Java Collection framework provides a **Stack class which models and implements Stack data Structure.**
 - The class is based on the **basic principle of last-in-first-out (LIFO).**
 - The class provides basic **operation push and pop.**
 - The class can also be referred to as the subclass of Vector.
- **Stack class has follow constructor:**
 - Stack()

UNIT – 2 Stack



Stack Data Structure

(Elements are added and removed from the top)

UNIT -2 Stack Class Methods

Method	Details
<i>Object push(object element)</i>	Pushes an element on the top of stack
<i>Object pop()</i>	Removes and returns the top element of the stack . An 'EmptyStackException' is thrown if we call pop() when the invoking stack is empty.
<i>Object peek()</i>	Returns the element on the top of the stack, but does not remove it.
<i>Boolean empty()</i>	It returns true if nothing is on the top of the stack. Else return false.
<i>Int search(object element)</i>	It determines whether an object exists in the stack. If the element is found, it returns the positions of the element from the top of the stack else return -1

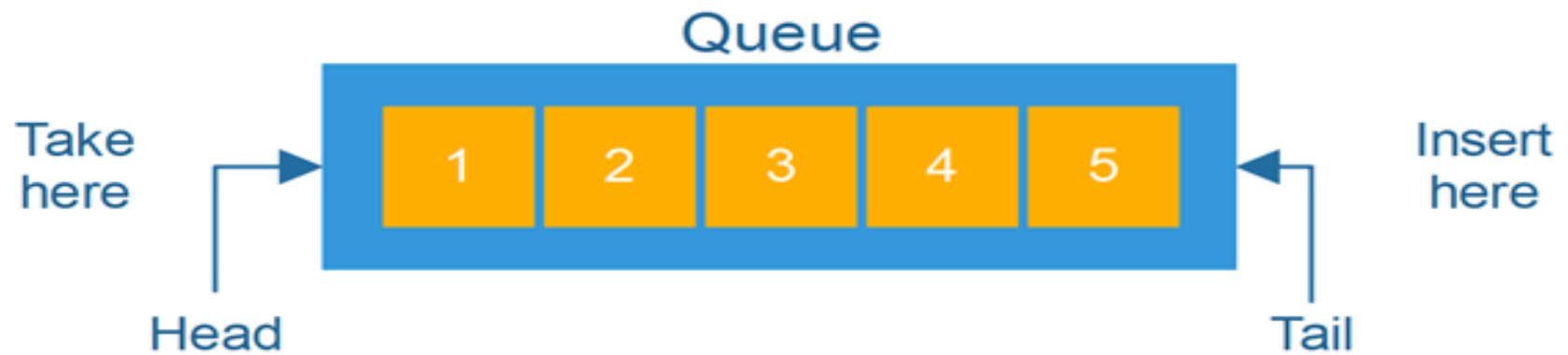
UNIT – 2 Stack

- Example:
 - Demostack.java

UNIT -2 Queue

- **Queue**
 - The Queue interface present in the java.utilpackage and extend the Collection interface.
 - **The class is based on the basic principle of last-in-first-out (FIFO).**
 - Being an interface the queue needs a concrete calss for the declaration, that are :
 - PriorityQueue
 - LinkedList
 - PriorityBlockingQueue

UNIT – 2 Queue



UNIT -2 Queue Interface Methods

Method	Details
<i>Boolean add(object)</i>	It is used to insert the specified element into queue and return true upon success.
<i>Boolean offer(object)</i>	It is used to insert the specified element into this queue.
<i>Objet remove()</i>	It is used to retrieves and removes the head of this queue.
<i>Object poll()</i>	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
<i>Object element()</i>	It is used to retrieves, but does not remove, the head of the queue.
<i>Object peek()</i>	It is used to retrieves, but does not remove, the head of this queue , or returns null if this queue is empty.

UNIT – 2 Queue

- Example:
 - Demoqueue.java

UNIT -2 SET

- SET
 - Set is a collection that **does not contain duplicates**. It extends the Collection interface.
 - **We can store at most one null value in Set.**
 - The **concept of union, intersection, and the difference of a set are available in the set interface** and supported by its subclasses
 - Set is implemented by HashSet, LinkedHashSet, and TreeSet.
- **Two classes under this interface**
 - **HashSet**
 - **TreeSet**
 - **LinkedHashSet**

UNIT -2 HashSet Class

- **HashSet**
 - HashSet class **implements the Set interface.**
 - It **does not guarantee that the order will remain constant** over time.
 - This class **permits the null element.**
 - It used for **storing the duplicate- free collection.**
 - For effectively storing and retrieving the elements but the order is not guaranteed by this class.
 - To retrieve the elements in a sorted order.
 - **Allows null values.**

UNIT -2 HashSet Class

- **HashSet**
 - **HashSet class has follow constructor:**
 - `Public HashSet()`
 - `Public HashSet(Collection C)`
 - `Public HashSet(int initialCapacity)`
- **Example:**
 - `Hashset1.java`

UNIT -2 TreeSet

- **TreeSet**
 - This class implements the ***Set* and *SortedSet*** interface.
 - It uses the tree to storage of its element.
 - It useful when **one needs to extract elements from a collection in a sorted manner.**
 - TreeSet offers a strict control over the order of elements in the collection. **The collection is a sorted collection.**
 - It may **not offer you the best performance** in terms of retrieving elements speedily.
 - **Does not permit null in the collection.**

UNIT -2 TreeSet Class Methods

Method	Details
<i>Comparator comparator()</i>	Returns the comparator used to order this sorted set , or null if this tree set uses its elements natural ordering
<i>Object first()</i>	Returns the first element currently in the sorted set
<i>Object last()</i>	Return the last element currently in the sorted set
<i>SortedSet tailSet(Object fromElement)</i>	Return a view of the portion of this set whose elements is greater than or equal to fromElement. The returned sorted set.
<i>SortedSet headSet(Object toElement)</i>	Return a view of the portion of this set this set whose elements are strictly less than toElement. The returned sorted set is backed by this set.

UNIT -2 TreeSet Class

- **TreeSet**
 - **TreeSet class has follow constructor:**
 - Public TreeSet()
 - Public TreeSet(Collection C)
 - Public TreeSet(Comparator C)
 - Public TreeSet(SortedSet S)
- Example:
 - Treetset1.java

UNIT -2 LinkedHashSet

- **LinkedHashSet**
 - This class **extends HashSet**.
 - **LinkeHashSet maintains linked list of the element in the set in the order in which they were inserted.**
 - That is, when cycling through a **LinkedHashSet** using an iterator, the elements will be returned in the order in which they were inserted.**Java LinkedHashSet class maintains insertion order.**
 - **This class permits the null element. Allows null values.**
 - **It used for storing the duplicate- free collection.**

UNIT -2 LinkedHasSet Class

- **LinkedHasSet**
 - **LinkedHasSet class has follow constructor:**
 - Public LinkedHasSet(int capacity)
 - Public LinkedHasSet(int capacity, float fillRatio)
- Example:
 - Demolinkedhasset.java

UNIT -2 MAPS Interface

- **MAPS Interface**

- A map contains values on the basis of key, i.e. key and value pair.
- Each key and value pair is known as an entry. A Map contains unique keys.
- A Map is useful if you have to search, update or delete elements on the basis of a key.
- MAPS Interface has following child
 - ***HashMap Class***
 - ***LinkedHashMap Class***
 - ***SortedMap Interface***
 - ***TreeMap Class***

UNIT -2 Maps Interface Methods

Method	Details
<i>Void clear()</i>	Remove all key value pairs from the invoking map
<i>Boolean containsKey(Object k)</i>	Returns “true” if the invoking map contains k as a key.
<i>Boolean containsValue(Object v)</i>	Returns “true” if the invoking map contains v as a value.
<i>Set entrySet()</i>	Return a set that contains the entries in the map. The set contains objects of type Map.Entry
<i>Boolean equals(object obj)</i>	Returns “true” if obj is a Map and contains the same entries

UNIT -2 Maps Interface Methods

Method	Details
<i>Object get(object k)</i>	Returns the value associated with the key k.
<i>Int hashCode()</i>	Returns the hash code for the invoking map.
<i>Boolean isEmpty()</i>	Returns “true” if the invoing map is empty.
<i>Set KeySet()</i>	Returns a Set that contains the keys in the invoking map.
<i>Object put(object k, object v)</i>	Puts an entry in the invoking map, overwritten any revious value associated with the key. The key and value are k and v respectively

UNIT -2 Maps Interface Methods

Method	Details
<i>Void putAll(Map m)</i>	Puts all the entries from m into this map
<i>Object remove(object k)</i>	Removes the entry whose key equals k
<i>Int Size()</i>	Returns the number of key-value paris in the map
<i>Collection values()</i>	Returns a collection containing the values in the map.

UNIT -2 HashMap Class

- **HashMap Class**

- Uses hashing as a technique to store key/value pairs so that the values can be searched efficiently according to the key.
- There order is not guaraanteed by HashMap.
- **HashMap allow null key and null value pairs to be stored.**
- **It is not an ordered collection** which means it does not return the keys and values in the same order in which they have been inserted into the HashMap.

UNIT -2 HashMap Class

- **HashMap Class**
 - **HashMap class has follow constructor:**
 - `Public HashMap()`
 - `Public HashMap(Map m)`
 - `Public HashMap(int initialCapacity)`
 - `Public HashMap(int initialCapacity, float loadFactor)`
- **Examples:**
 - `Hashmap1.java`
 - `Hashmap2.java`
 - `Hashmap3.java`

UNIT -2 TreeMap

- **TreeMap**
 - **TreeMap is implemented from SortedMap.**
 - This class guarantees that the map **will be in ascending key order**, sorted according to the natural order for the key's class.
 - **TreeMap contains sorted mapping of key/value pairs.**
 - **TreeMap Not allow null key and null value** pairs to be stored.
- **TreeMap class has follow constructor:**
 - `Public TreeMap()`
 - `Public TreeMap(Comparator c)`
 - `Public TreeMap(Map p)`
 - `Public TreeMap(SortedMap m)`

UNIT -2 TreeMap Class

- **TreeMap Class**
 - Examples:
 - Treemap1.java

UNIT -2 HashTable

- **HashTable**

- Like HashMap, Hashtable stores **key/value pairs** in a hash table.
- Java Hashtable class **contains unique elements**.
- Java Hashtable class **doesn't allow null key or value**.
- A Hashtable is **an array of a list**. Each **list is known as a bucket**. The position of the bucket is identified by calling the **hashCode() method**.
- A Hashtable contains **values based on the key**.

-

UNIT -2 HasTable Class

- **HashTable Class**
 - **HashTable class has follow constructor:**
 - Hashtable()
 - Hashtable(int size)
 - Hashtable(int size, float fillRatio)
 - Hashtable(Map m)
 - Examples:
 - DemoHashtable1.java
 - DemoHashTable.java

UNIT -2 Iterator

- **Iterator**
 - ‘Iterator’ is an **interface** which belongs to collection framework.
 - It **allows us to traverse the collection, access the data element and remove the data elements of the collection.**
 - we can **traverse a List or Set in forward direction.**
 - Before you can access a collection through an iterator, you must obtain one. **Each of the collection classes provides an iterator() method** that returns an iterator to the start of the collection.

UNIT -2 Iterator

Method	Details
<i>boolean hasNext()</i>	Returns true if there are more elements. Otherwise, returns false.
<i>Object next()</i>	Returns the next element . Throws NoSuchElementException if there is not a next element.
<i>void remove()</i>	Removes the current element . Throws IllegalStateException if an attempt is made to call <code>remove()</code> that is not preceded by a call to <code>next()</code> .

- Examples:
 - DemoIterator.java

UNIT -2 ListIterator

- **ListIterator**

- ‘ListIterator’ in Java is an Iterator which **allows users to traverse Collection in both direction.**
- It **extends Iterator interface.**
- It is **useful only for List implemented classes.**
- Unlike Iterator, It **supports all four operations: CRUD (CREATE, READ, UPDATE and DELETE).**
- Unlike Iterator, It **supports both Forward Direction and Backward Direction iterations.**
- It is a **Bi-directional Iterator.**

UNIT -2 ListIterator

Method	Details
<i>void add(Object obj)</i>	Inserts obj into the list in front of the element that will be returned by the next call to next().
<i>boolean hasNext()</i>	Returns true if there is a next element. Otherwise, returns false.
<i>boolean hasPrevious()</i>	Returns true if there is a previous element. Otherwise, returns false.
<i>Object next()</i>	Returns the next element. A NoSuchElementException is thrown if there is not a next element.
<i>int nextIndex()</i>	Returns the index of the next element. If there is not a next element, returns the size of the list.

UNIT -2 ListIterator

Method	Details
<i>Object previous()</i>	Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
<i>int previousIndex()</i>	Returns the index of the previous element. If there is not a previous element, returns -1.
<i>void remove()</i>	Removes the current element from the list. An IllegalStateException is thrown if remove() is called before next() or previous() is invoked.
<i>void set(Object obj)</i>	Assigns obj to the current element. This is the element last returned by a call to either next() or previous()

Examples:
DemoListIterator.java



UNIT 2 COMPLETED