Part-3

1) **Exception**
   - **a)Define & RealTime example**
   - **b)Types of Exception and explain**
   - **c)Exception handling**
   - **d)Explain try,catch –rules,finally**
   - **e)throw**

     1.1.Define
     1.2.Syntax
     1.3.Note

   - **f)throws**

     1.1.Define
     1.2.Note
     1.3.Syntax
     1.4.Rules

   - **g)Difference between Throw and Throws**
   - **h)Difference b/w final, finally ,finalize**
   - **i)Throwabletype of object ref.printStackTrace()**
   - **j)Object Propagation**
   - **k)Unchecked Exception propagation**
   - **l)Custom Exception**

     1.1.Define
     1.2.We can create custom exception for both checked and unchecked

Feature	List	Set
Type	Interface	Interface
Package	`java.util`	`java.util`
Duplicates	✓ Allowed	✗ Not allowed
Order	✓ Maintains insertion order	✗ No order (except LinkedHashSet, TreeSet)
Indexing	✓ Supports index	✗ No index
Access	`get(index)`	No direct access
Sorting	✓ Can be sorted	✗ Cannot be sorted directly
Null elements	✓ Multiple nulls (ArrayList)	✗ Only one null (HashSet)
Performance	Good for search by index	Good for uniqueness
Iterator	ListIterator + Iterator	Only Iterator
Implementations	ArrayList, LinkedList, Vector	HashSet, LinkedHashSet, TreeSet

Feature	ArrayList	LinkedList	Vector	Stack
Data structure	Resizable array	Doubly linked list	Resizable array	LIFO (Vector)
Thread-safe	✗	✗	✓	✓
Access by index	O(1)	O(n)	O(1)	O(1)
Insert/delete middle	O(n)	O(1)	O(n)	O(n)
Implements	List	List, Deque	List	Vector
Legacy	✗	✗	✓	✓
Use case	Fast search	Frequent insertion/deletion	Thread-safe array	LIFO operations

Feature/Aspect	ArrayList	LinkedList	Vector	Stack
Underlying Data Structure	Dynamic array	Doubly linked list	Dynamic array	**Extends Vector** (dynamic array)
Implements Interfaces	List, RandomAccess, Cloneable, Serializable	List, Deque, Cloneable, Serializable	List, RandomAccess, Cloneable, Serializable	List, RandomAccess, Cloneable, Serializable
Insertion/Deletion	Slow (except at end, O(1) amortized)	Fast (O(1) at start or end, O(n) for index access)	Slow (like ArrayList)	Same as Vector (slow in middle)
Access by Index	Fast (O(1))	Slow (O(n))	Fast (O(1))	Fast (O(1))
Thread Safety	Not synchronized	Not synchronized	Synchronized (thread-safe)	Synchronized (thread-safe)
Usage	General-purpose list, frequent access	Frequent insert/delete, queue/deque operations	Legacy code, thread-safe list	LIFO operations (push/pop)
Memory Overhead	Low	Higher (due to node pointers)	Low	Low (inherits Vector overhead)
Performance	Fast for random access	Fast for insertion/deletion	Slightly slower due to synchronization	Same as Vector

Feature/Aspect	HashSet	LinkedHashSet	TreeSet
Underlying Data Structure	Hash table	Hash table + doubly linked list	Red-Black tree (balanced binary search tree)
Implements Interfaces	Set, Cloneable, Serializable	Set, Cloneable, Serializable	Set, NavigableSet, SortedSet, Cloneable, Serializable
Order of Elements	No order (unordered)	Insertion order	Sorted (natural order or via Comparator)
Allows Null	Yes (only one null)	Yes (only one null)	No
Performance (Add/Remove/Contains)	O(1) average	O(1) average	O(log n)
Iteration Order	Unpredictable	Predictable (insertion order)	Sorted order
Memory Overhead	Low	Slightly higher (due to linked list)	Higher (tree structure)
Thread Safety	Not synchronized	Not synchronized	Not synchronized
Usage	Fast operations without order	Maintain insertion order	Maintain sorted order

Feature/Aspect	HashMap	LinkedHashMap	TreeMap
Underlying Data Structure	Hash table	Hash table + doubly linked list	Red-Black tree (balanced binary search tree)
Implements Interfaces	Map, Cloneable, Serializable	Map, Cloneable, Serializable	Map, NavigableMap, SortedMap, Cloneable, Serializable
Order of Entries	No order (unordered)	Insertion order	Sorted order (natural or via Comparator)
Allows Null	Yes (1 null key, multiple null values)	Yes (1 null key, multiple null values)	No null keys, allows multiple null values
Performance (Put/Get/Remove)	O(1) average	O(1) average	O(log n)
Iteration Order	Unpredictable	Predictable (insertion order)	Sorted order
Memory Overhead	Low	Slightly higher (due to linked list)	Higher (tree structure)
Thread Safety	Not synchronized	Not synchronized	Not synchronized
Usage	Fast key-value operations without order	Maintain insertion order	Maintain sorted keys; range queries

---

Feature / Aspect	Set	Map
Type	Collection of elements	Collection of key–value pairs
Package	`java.util`	`java.util`
Extends / Interface	Extends `Collection`	Does **NOT** extend `Collection`
Data Storage	Stores **only values**	Stores **key → value** pairs
Duplicates	❌ Does not allow duplicate elements	❌ Does not allow duplicate **keys** (values can be duplicate)
Null Allowed	Depends on implementation (e.g., HashSet allows one null)	Depends on implementation (e.g., HashMap allows one null key, multiple null values)
Access Mechanism	Access via **iteration only**	Access via **keys** using `get(key)`
Index-Based Access	❌ No indexing, no `get()` method	❌ No index, but key-based access
Order Maintenance	Depends on implementation (LinkedHashSet, TreeSet)	Depends on implementation (LinkedHashMap, TreeMap)
Common Implementations	HashSet, LinkedHashSet, TreeSet	HashMap, LinkedHashMap, TreeMap, Hashtable
Use Case	When you need **unique elements only**	When you need **key–value mapping**

How LinkedHashSet Stores Elements