Introduction To Java

LESSON 5

Learning Outcomes : Lesson 5

- Generic
- Sorting Collection
  - Comparable Interface
  - Comparator Interface
- Regular Expression
- Exception Handling
- Collections

Generic

- Generics means parameterized types. The idea is to allow type (Integer, String, … etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.
- An entity such as class, interface, or method that operates on a parameterized type is called generic entity.

Generic

```java
class Test<T>
{
    // An object of type T is declared
    T obj;
    Test(T obj) {  this.obj = obj;  }  // constructor
    public T getObject()  { return this.obj; }
}

// Driver class to test above
class Main
{
    Run | Debug
    public static void main (String[] args)
    {
        // instance of Integer type
        Test <Integer> iObj = new Test<Integer>(15);
        System.out.println(iObj.getObject());

        // instance of String type
        Test <String> sObj =
                    new Test<String>("Introduction of Java");
        System.out.println(sObj.getObject());
    }
}
```

Sort List of Object Type

- We generally use Collections.sort() method to sort a simple array list ex. The list of String type.

- If the ArrayList is of custom object type ex the ArrayList of Student type then in such case you have two options for sorting.
  - Comparable Interface
  - Comparator Interface

Comparable Interface

- Java Comparable interface is used to order the objects of the user-defined class.
- This interface is found in java.lang package and contains only one method named compareTo(Object).
- It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be name, age or anything else.

CompareTo(object) Method

- It is used to compare the current object with the specified object. It returns :
  - positive integer, if the current object is greater than the specified object.
  - negative integer, if the current object is less than the specified object.
  - zero, if the current object is equal to the specified object.

Comparator Interface

- Java Comparator interface is used to order the objects of a user-defined class.
- This interface is found in java.util package and contains one method compare(Object obj1,Object obj2)
- It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example name, age or anything else
- Compare method is used to compare the first object with the second object.

Regular Expression

- A regular expression defines a search pattern for strings. The abbreviation for regular expression is regex.
- The search pattern can be anything from a simple character, a fixed string or a complex expression containing special characters describing the pattern.
- The process of analyzing or modifying a text with a regex is called: The regular expression is applied to the text/string. The pattern defined by the regex is applied on the text from left to right.

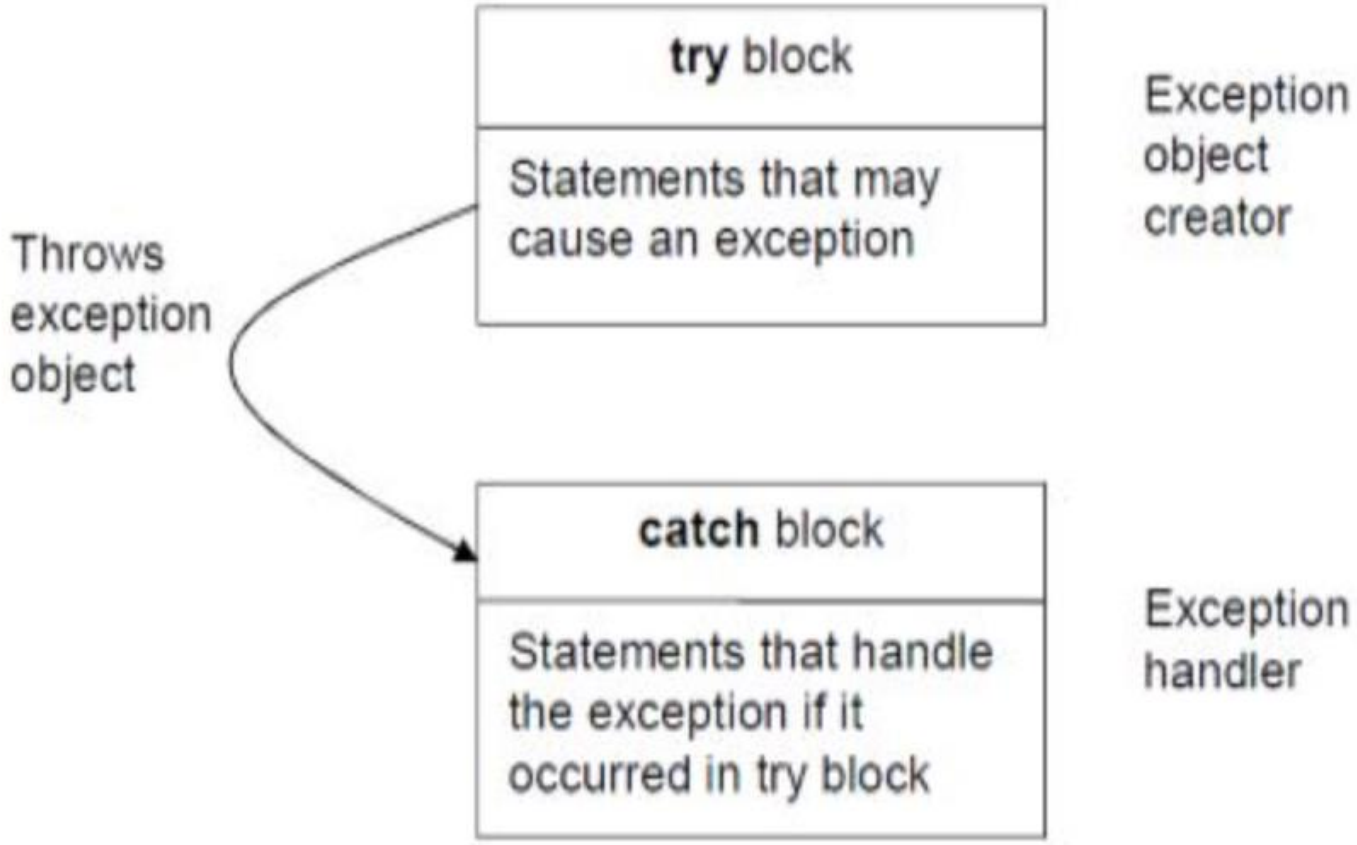
```
List<String> zips = new ArrayList<String>();

//Valid ZIP codes
zips.add("K1A 0B1");
zips.add("B1Z 0B9");

//Invalid ZIP codes
zips.add("K1A 0D1");
zips.add("W1A 0B1");
zips.add("Z1A 0B1");

String regex = "^(?!.*[DFIOQU])[A-VXY][0-9][A-Z] ?[0-9][A-Z][0-9]$";

Pattern pattern = Pattern.compile(regex);

for (String zip : zips)
{
    Matcher matcher = pattern.matcher(zip);
    System.out.println(matcher.matches());
}

Output:

true
true

false
false
```

Exception in Java

- Exception in Java is an event that interrupts the execution of program instructions and disturbs the normal flow of program execution. It is an object that wraps an error event information that occurred within a method and it is passed to the runtime system. In Java, exceptions are mainly used for indicating different types of error conditions.
- There are two types of errors:
  - Compile time error
  - Run time error
- A Runtime error is called an Exceptions error. It is any event that interrupts the normal flow of program execution. Example for exceptions are, arithmetic exception, Nullpointer exception, Divide by zero exception, etc.

Try Catch Block

- Java provides an inbuilt exceptional handling.
- The normal code goes into a TRY block. The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.
- The exception handling code goes into the CATCH block. A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.


try block
Statements that may cause an exception
Exception object creator
Throws exception object
catch block
Statements that handle the exception if it occurred in try block
Exception handler

Example of Try Catch Block

- If an exception occurs in try block then the control of execution is passed to the corresponding catch block. A single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last.

  The generic exception handler can handle all the exceptions but you should place is at the end, if you place it at the before all the catch blocks then it will display the generic message. You always want to give the user a meaningful message for each type of exception rather than a generic message.

Types of Exception in Java

- **ArithmeticException :** It is thrown when an exceptional condition has occurred in an arithmetic operation.
- **ArrayIndexOutOfBoundsException :** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
- **NullPointerException :** This exception is raised when referring to the members of a null object. Null represents nothing
- **StringIndexOutOfBoundsException :** It is thrown by String class methods to indicate that an index is either negative than the size of the string
- **ClassNotFoundException :** This Exception is raised when we try to access a class whose definition is not found

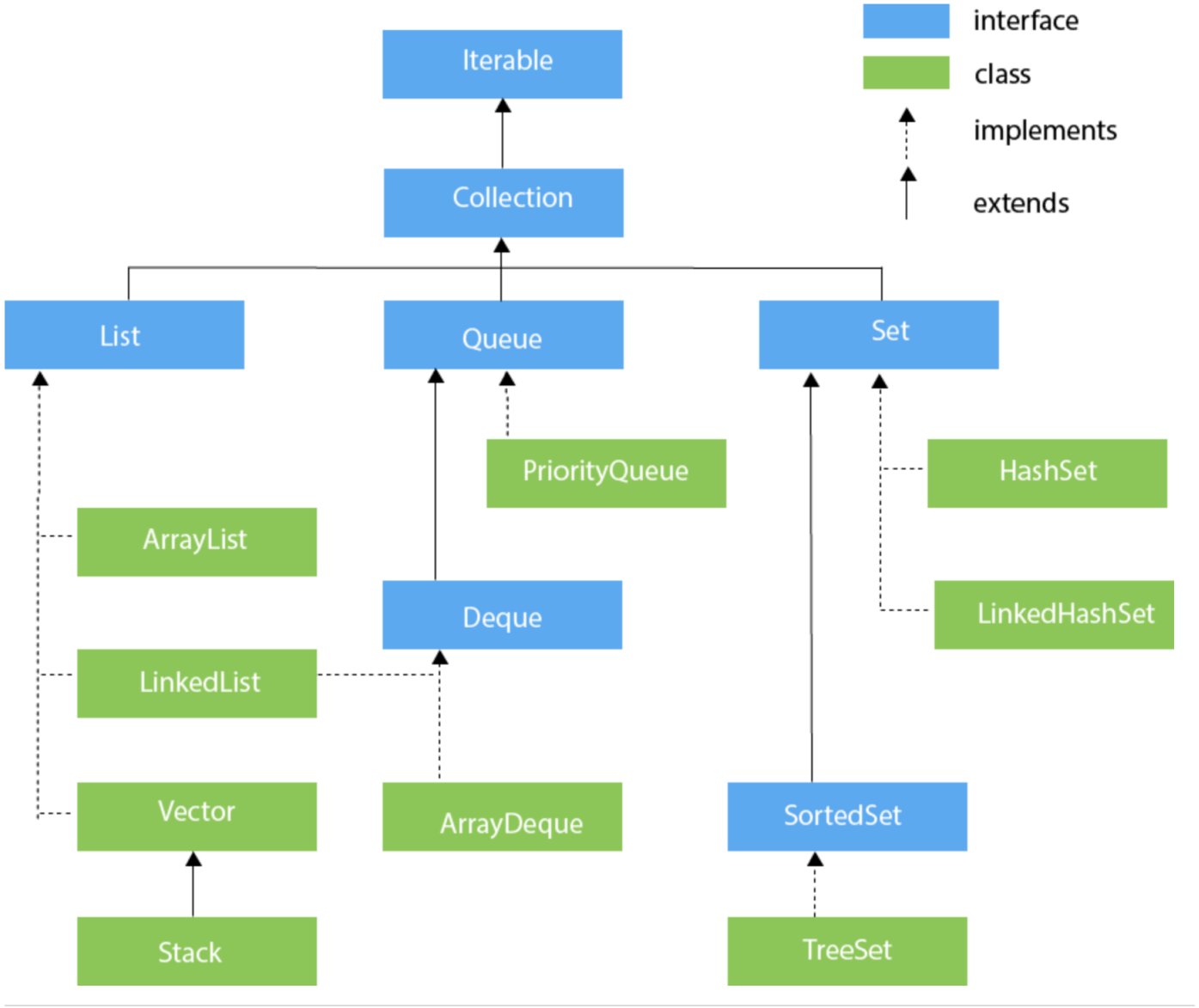
Finally Block

- A finally block contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.
- A finally block must be associated with a try block, you cannot use finally without a try block.
- In normal case, when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
- The statement in Finally block execute even if the try block contains control transfer statements like return, break or continue.

Collections

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Hierarchy of Collection Framework



Methods of Collection Interface

- **public boolean add(E e)** : It is used to insert an element in this collection.
- **public boolean addAll(Collection<? extends E> c)** : It is used to insert the specified collection elements in the invoking collection.
- **public boolean remove(Object element)** : It is used to delete an element from the collection.
- **public boolean removeAll(Collection<?> c)** : It is used to delete all the elements of the specified collection from the invoking collection.

Methods of Collection Interface

- **public int size()** : It returns the total number of elements in the collection.
- **public void clear()** : It removes the total number of elements from the collection.
- **public boolean contains(Object element)** : It is used to search an element.
- **public Object[] toArray()** : It converts collection into array.
- **public boolean isEmpty()** : It checks if collection is empty.
- **public boolean equals(Object element)** : It matches two collections.

List Interface

- List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.
- List interface is implemented by the classes
  - ArrayList
  - Vector
  - Stack.
  - LinkedList
- There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

Methods of List Interface

- **Add(int index, element)** : This method is used to add an element at a particular index in the list.
- **Remove(int index)** : This method removes an element from the specified index.
- **Get(int index)** : This method returns element at the specified index.
- **Set(int index, element)** : This method replaces element at given index with new element.
- **IndexOf(element)** : This method returns the first occurrence of the given element or *-1* if the element is not present in the list.
- **LastIndexOf(element)** : This method returns the last occurrence of the given element or *-1* if the element is not present in the list.

Classes Which Implements the List Interface

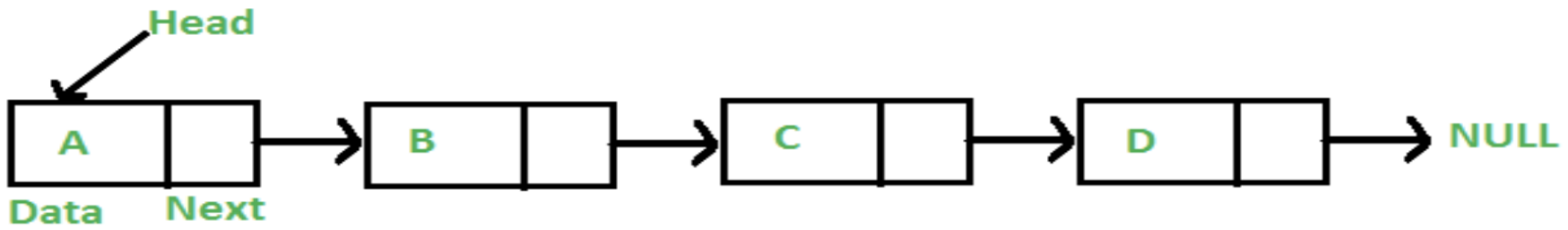
- **ArrayList** : ArrayList class which is implemented in the collection framework provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
  - ArrayList<T> list = new ArrayList<T>();
- **Vector** : Vector is a class which is implemented in the collection framework implements a growable array of objects. Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
  - Vector<T> v = new Vector<T>();

Classes Which Implements the List Interface

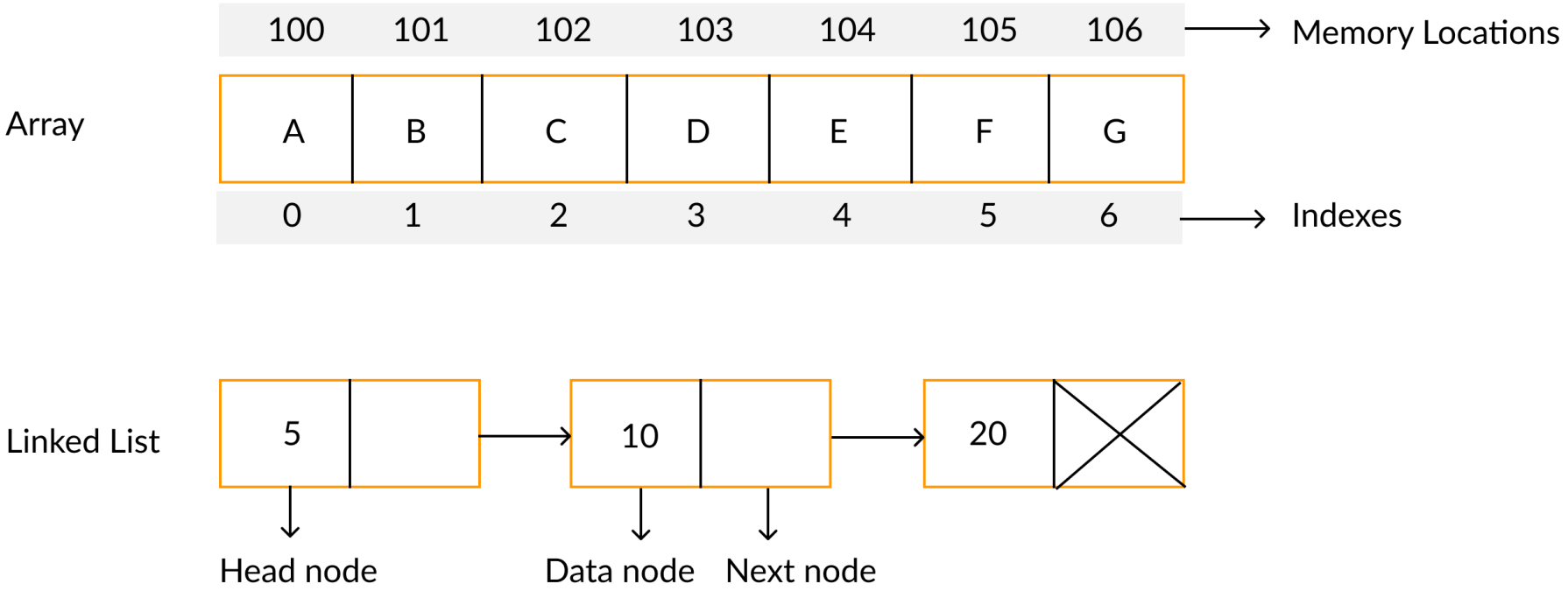
- **Stack** : Stack is a class which is implemented in the collection framework and extends the vector class models and implements the Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek.
- List<T> stack = new Stack<T>();
  - What is Stack Data Structure ? Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottom most position remains in the stack for the longest period of time.

Classes Which Implements the List Interface

- **LinkedList** : LinkedList is a class which is implemented in the collection framework which inherently implements the linked list data structure. It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.
- List<Integer> linkedList = **new** LinkedList<Integer>();

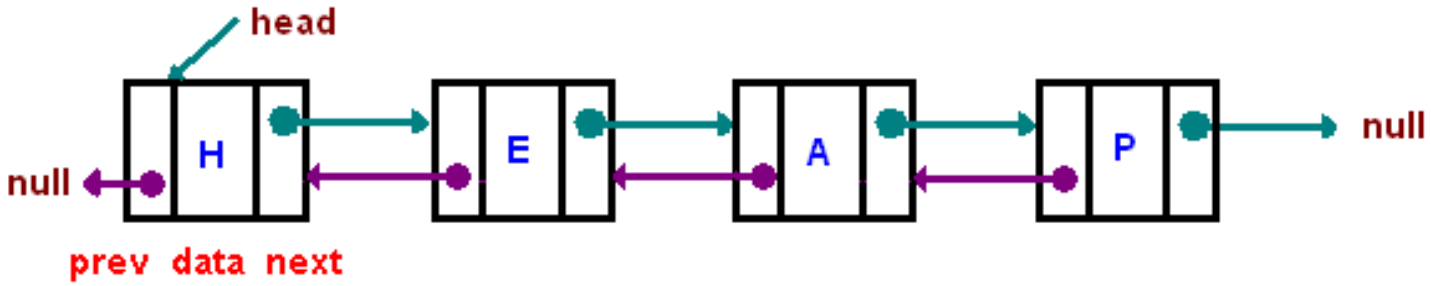


LinkedList vs Array



LinkedList vs ArrayList vs Vector

- ArrayList and Vector both grow and shrink dynamically to maintain optimal use of storage but the way they resize is different. ArrayList increments 50% of the current array size if the number of elements exceeds its capacity, while vector increments 100% essentially doubling the current array size. LinkedList internally uses a **doubly linked list** to store the elements.



- Linkedlist also implements queue interface which adds more methods than arraylist and vector, such as offer(), peek(), poll(), etc.

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.

Iterate in Java

- 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.
- java.util package has public interface Iterator and contains three methods:
  - **boolean hasNext()**: It returns true if Iterator has more element to iterate.
  - **Object next()**: It returns the next element in the collection until the hasNext()method return true. This method throws 'NoSuchElementException' if there is no next element.
  - **void remove()**: It removes the current element in the collection. This method throws 'IllegalStateException' if this function is called before next( ) is invoked.

```java
// Java code to illustrate the use of iterator
import java.io.*;
import java.util.*;

class Test {
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<String>();

        list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");

        // Iterator to traverse the list
        Iterator iterator = list.iterator();

        System.out.println("List elements : ");

        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");

        System.out.println();
    }
}
```

Queue Interface

- Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.
- FIFO real example is like a queue of people at ticket-window: The person who comes first gets the ticket first. The person who is coming last is getting the tickets in last. Therefore, it follows first-in-first-out (FIFO) strategy of queue.

Set Interface

- Set Interface in Java is present in java.util package. It extends the Collection interface.
- It represents the unordered set of elements which doesn't allow us to store the duplicate items.
- Set is implemented by
  - HashSet
  - LinkedHashSet
  - TreeSet.

Set Interface vs List Interface

- The first difference between the Java Set and List interface is, that the same element cannot occur more than once in a Java Set. This is different from a Java List where each element can occur more than once.
- The second difference between a Java Set and Java List interfaces is, that the elements in a Set has no guaranteed internal order. The elements in a List has an internal order, and the elements can be iterated in that order.

HashSet

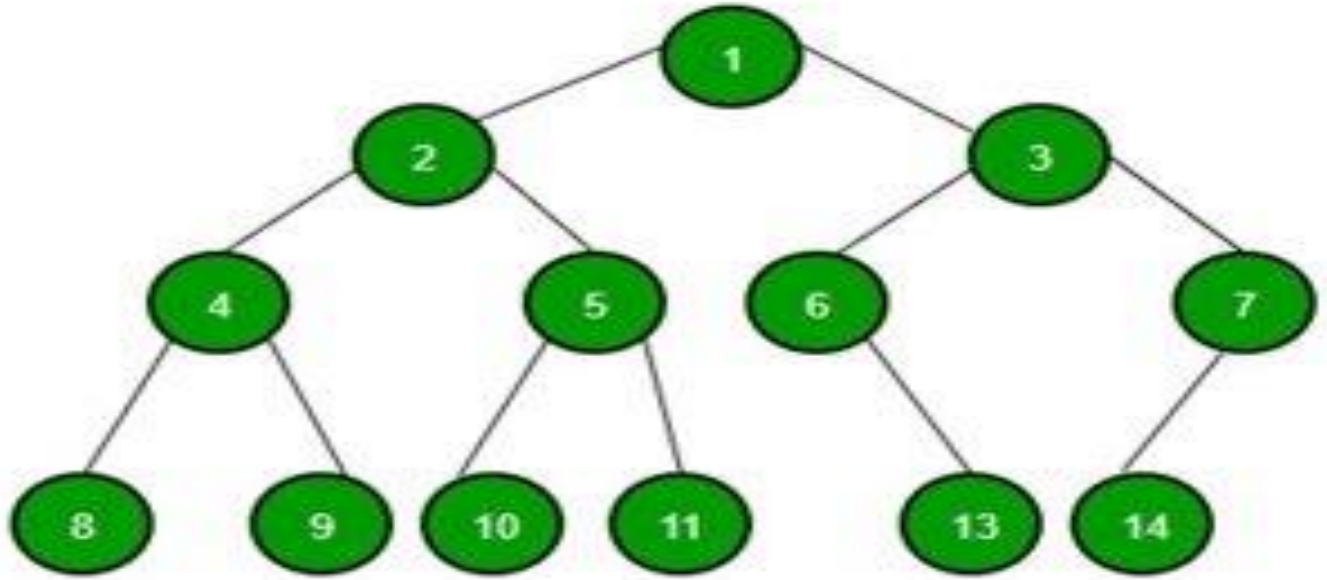
- The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance.
- No guarantee is made as to the iteration order of the set which means that the class does not guarantee the constant order of elements over time.
- NULL elements are allowed in HashSet.
- As it implements the Set Interface, duplicate values are not allowed.
- Set <T> set = new HashSet<T> ()

TreeSet

- TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage.
  - SortedSet interface contains the methods inherited from the Set interface and adds a feature that stores all the elements in this interface to be stored in a sorted manner.
- The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided.
- TreeSet<T> ts = new TreeSet<T>()
- We need to keep a note that duplicate elements are not allowed and all the duplicate elements are ignored.
- Null values are not accepted by the TreeSet.
- Threre are two more methods in TreeSet. **First()** method will return first element in TreeSet and **Last()** method return last element in TreeSet

Binary Tree Data Structure

- A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



Map Interface

- The Map interface present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.
- A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value like the HashMap and LinkedHashMap, but some do not like the TreeMap.
- There are two interfaces for implementing Map in java Map and SortedMap, and three classes: HashMap, TreeMap and LinkedHashMap.
- Maps are perfect to use for key-value association mapping. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys.

HashMap

- HashMap is a part of Java's collection since Java 1.2. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs.
- HashMap<String, Integer> map = **new** HashMap<>();
- HashMap doesn't allow duplicate keys but allows duplicate values. That means A single key can't contain more than 1 value but more than 1 key can contain a single value.
- HashMap allows null key also but only once and multiple null values.
- Using HashMap allows insertion of key value pair so It helps to have a faster access of elements due to hashing technology.

Methods of HashMap Class

- **Object put(Object key, Object value)** : It is used to insert a particular mapping of key-value pair into a map.
- **Boolean containsKey(Object key)** : Used to return True if for a specified key, mapping is present in the map.
- **Boolean containsValue(Object value)** : Used to return true if one or more key is mapped to a specified value.
- **Object get(Object key)** : It is used to retrieve or fetch the value mapped by a particular key.
- **Object remove(Object key)** : It is used to remove the values for any particular key in the Map.
- **Set keySet()** : It is used to return a set view of the keys.
- **Set entrySet()** : It is used to return a set view of the hash map.

Iterate through HashMap

- Using Foreach to iterate through the hashMap which is advised approach because we can have a full control on key and value of HashMap.
- In each for cycle, a new key-value couple is assigned to the pair variable.

```
import java.util.HashMap;
import java.util.Map;

public class IterateHashMap {

    public static void main(String[] args) {
        Map<String, String> map = new HashMap<String, String>();
        map.put("key1", "value1");
        map.put("key2", "value2");

        for (Map.Entry<String, String> entry : map.entrySet()) {
            System.out.println(entry.getKey() + " = " + entry.getValue());
        }
    }
}
```