Collections & JDK8

(Genie Ashwani)

Genie Ashwani

SPARK 3.0 Batch

Contents

Collection Framework

Introduction:

- Programming languages and Technologies are used to develop applications.
- Applications are used in communication.
- Applications store and process information.

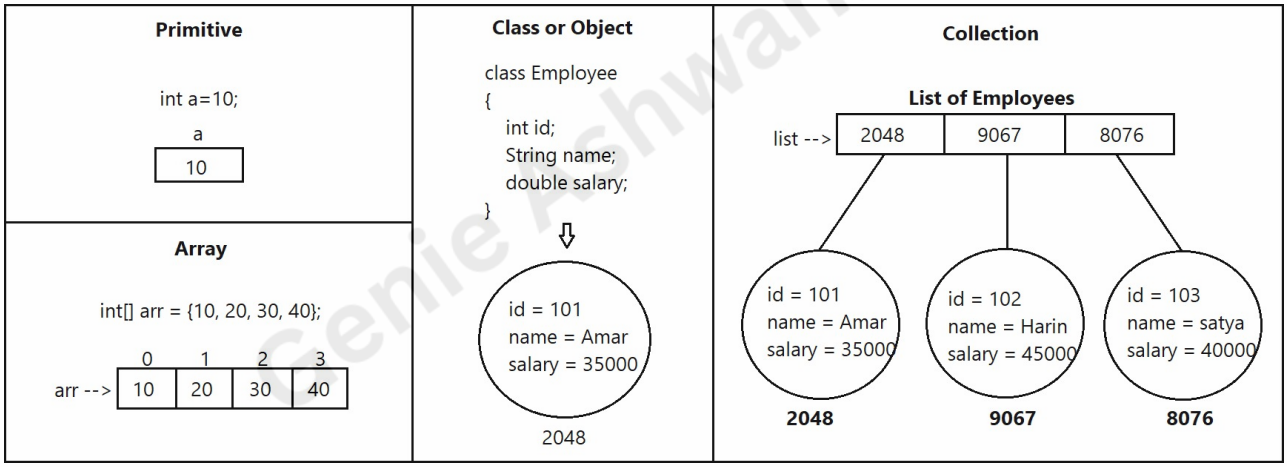
For example, a Bank Application –

Store customer's information and transactions information.

Customers use Bank application to communicate with Bank people and other account holder to perform the transactions.

How to store information in application?

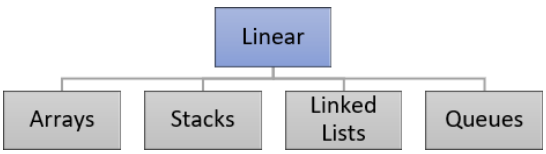
- We use variables of different data types to store information.
  - **Primitive type:** Store only one value at a time.
  - **Array:** Store more than one value but of same type
  - **Object:** Store more than one value of different types
  - **Array of Objects :** Store fixed number of objects(Not recommended)
  - **Collection of Objects:** Store objects dynamically(Not fixed size).



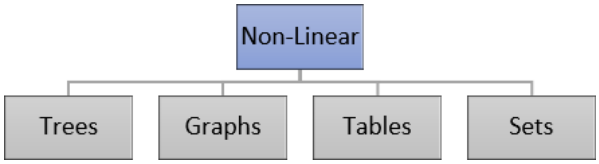
What are Data Structures? Use?

- Data structures are used to organize the data.
- We can perform operations quickly and easily on organized data.
- Data structures either Linear or Non-Linear.

Linear Data Structures: arrange the data sequentially in which elements are connected to its previous and next adjacent.



Non-Linear Data Structures: in which one element connected to multiple elements and the elements arranged in two-dimensional or multi-dimensional format.



Define Collection?

- Collection is a group of objects.
- Examples, List, Set, Queue, Map etc.

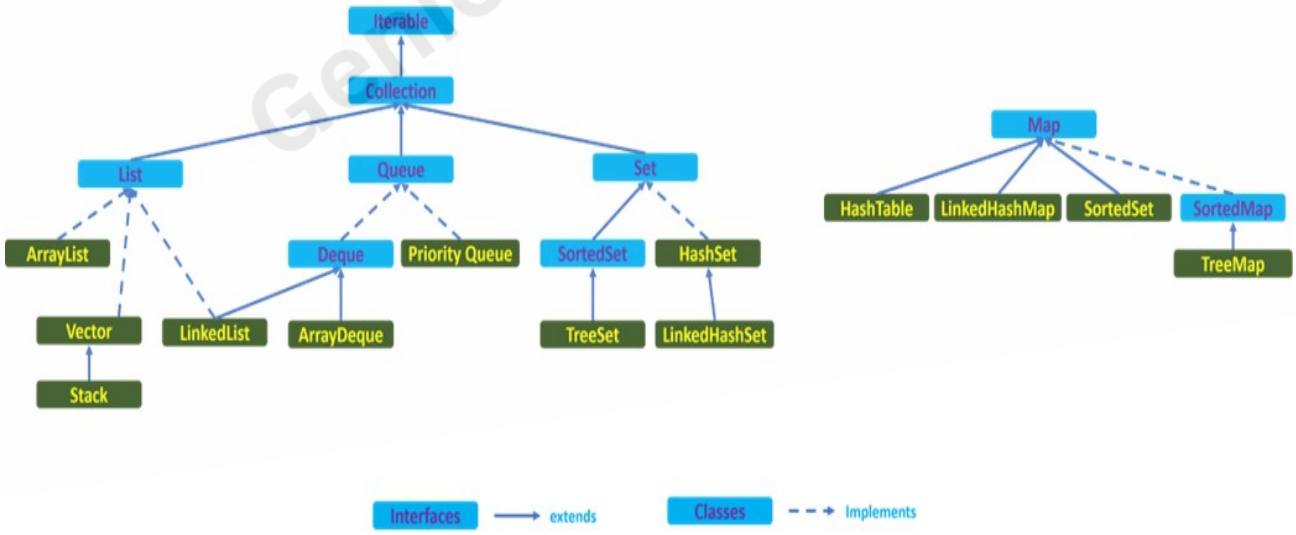
What is Collection framework?

- Collection Framework is a collection of interfaces and implemented classes.
- Collection Framework provides implementations of Data structures & Algorithms by which we can store and process information without implementing them.

What is the need of storing group of objects?

- To store the record type information which is fetching from Database.
- To perform Different types of operations on group of objects like insertion, Deletion, Updating, searching, sorting etc...
- Can set multiple objects to method as a parameter.
- Method can return multiple objects at a time after processing.

Collection Hierarchy: The following diagram represents interfaces and classes available in java Collection Framework



Define Array and Collection:

- Array is static – fixed size
- Collection is dynamic – size grows and shrinks with insertions and deletions.

Define List, Set and Map:

List	Set	Map
List is index based.	Set is not index based.	Map is not index based.
List allow duplicates.	Set doesn't allow duplicates.	Map store elements using keys. Keys must be unique. Elements can be duplicated.

List implemented by:

1. **ArrayList**
    a. Accessing elements much faster.
    b. Insertions and Deletions are slower – shifting elements takes time.

2. **Vector:**
    a. Accessing elements much faster.
    b. Insertions and Deletions are slower – shifting elements takes time.

3. **Stack:**
    a. Stack follows Last In First Out (LIFO) rule.
    b. Inserting and removing elements from one end called TOP.

4. **LinkedList:**
    a. Accessing element slower, nodes-based access.
    b. Insertions and Deletions are faster – No shifting of elements.

Set implemented by:

1. **HashSet:** doesn't maintain insertion order.
2. **LinkedHashSet:** maintains insertion order.
3. **TreeSet:** maintains sorted order.

Map implemented by:

1. **Hashtable:** maintains sorted order using keys. Null keys not allowed.
2. **HashMap:** doesn't maintain insertion order. One null key allowed.
3. **LinkedHashMap:** maintain insertion order. One null key allowed.
4. **TreeMap:** maintain sorted order using keys. Null keys not allowed.

Queue implemented by:

PriorityQueue: It is not an order collection and allow duplicates**.** Priority queue elements are retrieved in sorted order. Head of the priority queue will be the smallest element. Once this element is retrieved, the next smallest element will be the head of the queue.

Genie Ashwani

Wrapper classes in Collections

Wrapper classes:

- Collection stores only objects (not primitive data).
- Wrapper classes providing functionality to perform conversions like
  - Primitive -> Object (Boxing)
  - Object -> Primitive (Un boxing)
- These conversions become automated since JDK5

Note: for every primitive type there is a wrapper class in java

Primitive type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Boxing: Conversion of primitive type into object type

```
int x = 10;
Integer obj = new Integer(x);
```

Un boxing: Conversion of object type into primitive type

```
int x = obj.intValue();
```

Auto Boxing: Auto conversion of boxing

```
int x = 10;
Integer obj = x;
```

Auto Un boxing: Auto conversion process of un boxing.

```
int x = obj;
```

Genie Ashwani

Generics

Generics:

- As we know, collection only store objects.
- Generics introduced in JDK5.
- Generics are used to specify what type of objects allowed to store into Collection.

Collection without Generics: Allow to store any type of Objects.

Syntax:

```
Collection c = new Collection();
c.add(10);
c.add(23.45);
c.add("java");
```

Collection with Generics: Allow only specific type of data Objects.

Syntax:

```
Collection<Integer> c = new Collection<Integer>();
c.add(10);
c.add("java");  // Error :
```

Collection with Generics that allows any type of object:

Syntax:

```
Collection<Object> c = new Collection<Object>();
c.add(10);
c.add("java");
c.add(23.45);
```

Note: Object is the super class of all classes in Java

If we store information in Object form, we need to downcast the object into corresponding type to perform operations.

For Example,

```
Collection<Object> c = new Collection<Object>();
c.add(10);
```

Downcast to Integer:

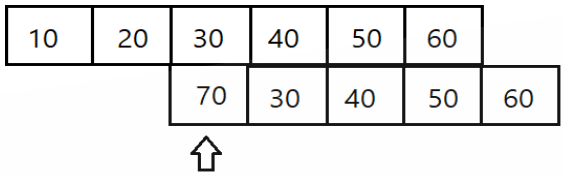
```
Integer x = c.get(0);
```

Genie Ashwani

ArrayList Collection

ArrayList:

- ArrayList is an ordered collection and allow duplicates.
- ArrayList is index based.
- Processing elements much faster (index based)
- Insertions and Deletions are slower (shifting of elements takes time)



Inserting 70 shifts all elements to right side

Methods:

Name	Description
int size()	Returns the number of elements in this list.
boolean add(E e)	Appends the specified element to the end of this list
Object remove(int index)	Removes the element at the specified position in this list
void clear()	Removes all of the elements from this list
void add(int index, E element)	Inserts element at the specified position in this list
Object get(int index)	Returns the element at the specified position in this list
boolean isEmpty()	Returns true if this list contains no elements
Object set(int index, E element)	Replaces the element at the specified position in this list with the specified element
boolean contains(Object o)	Returns true if this list contains the specified element
int indexOf(Object o)	Returns the index of the first occurrence of the specified element, or -1 if this list does not contain the element.
Iterator<Object> iterator()	Returns an iterator over the elements in this list
boolean addAll(Collection c)	Appends all of the elements in the specified collection to the end of this list
Object clone()	Returns a shallow copy of this ArrayList instance
ListIterator listIterator(int index)	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
Object[] toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).

Genie Ashwani

Program to display ArrayList and its size:

- add() method is used to append element to the list.
- size() method returns the length of list.

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		List<Integer> list = new ArrayList<Integer>();
		list.add(10);
		list.add(20);
		list.add(30);
		list.add(40);
		list.add(50);
		System.out.println("List is : " + list);
		System.out.println("Size is : " + list.size());
	}
}
```

Program to check the list is empty or not:

- isEmpty() method returns true if the list doesn't contains elements else returns false

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		List<Integer> list = new ArrayList<Integer>();
		if(list.isEmpty())
			System.out.println("List is empty");
		else
			System.out.println("List contains elements");
	}
}
```

Program to display the element of specified index:

- get(int index) returns the element of specified index.

```
import java.util.*;
class Code
{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
```

```
                List<Integer> list = new ArrayList<Integer>();
                for (int i=10 ; i<=50 ; i+=10){
                        list.add(i);
                }
                System.out.println("List is : " + list);
                System.out.println("Enter index to display value : ");
                int loc = sc.nextInt();
                System.out.println("Element @ index-" + loc + " is : " + list.get(loc));
        }
}
```

We specify the error message – if the index value is not present:

```
if(loc>=0 && loc<=list.size()-1){
        System.out.println(list.get(loc));
}
else{
        System.out.println("Invalid index");
}
```

```
try{
        System.out.println(list.get(loc));
}
catch(IndexOutOfBoundsException e){
        System.out.println("Invalid
        index");
}
```

Insert element into specified index: add(int index, E e) method is used to insert element into specified index.

Instructions to code:

- Create ArrayList with 5 elements 10, 20, 30, 40, 50
- Read index to insert.
- Check whether the index is present or not
- If the index is present, then read the value and insert.
- If the index is not present, display Error message.

```
import java.util.*;
class Code
{
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                List<Integer> list = new ArrayList<Integer>();
                list.add(10);
                list.add(20);
                list.add(30);
                list.add(40);
```

```
                list.add(50);
                System.out.println("List is : " + list);
                System.out.print("Enter index to insert : ");
                int loc = sc.nextInt();
                if(loc>=0 &&  loc<list.size()){ System.out.print("Enter
                        element to insert : ");int ele = sc.nextInt();
                        list.add(loc, ele);
                        System.out.println("List is : " + list);
                }
                else{
                        System.out.println("Invalid index");
                }
        }
}
```

Program to remove all elements from the list: clear() method removes all elements from the list.

Instructions to code:

- Create list with 5 elements.
- Display – List is not empty
- Remove all elements using clear() method
- Display – List is empty.

```
import java.util.*;
class Code
{
        public static void main(String[] args){
                List<Integer> list = new ArrayList<Integer>();
                for(int i=1 ; i<=5 ; i++)
                        list.add(i);

                System.out.println("List is : " + list);
                if(list.isEmpty())
                        System.out.println("List is empty");
                else
                        System.out.println("List is not empty");
```

```
                list.clear();
                System.out.println("List is : " + list);
                if(list.isEmpty())
                        System.out.println("List is empty");
                else
                        System.out.println("List is not empty");
        }
}
```

Program to remove index element: remove(int index) method removes element of specified index.

Instructions to code:

- Create list with elements
- Read index value.
- If the index is valid – remove the element and display list
- If the index is not valid – display error message.

```
import java.util.*;
class Code
{
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                List<Integer> list = new ArrayList<Integer>();
                for(int i=1 ; i<=5 ; i++){
                        list.add(i);
                }
                System.out.println("List is : " + list);
                System.out.print("Enter index to remove : ");
                int loc = sc.nextInt();
                if(loc>=0 && loc<list.size()){
                        list.remove(loc);
                        System.out.println("List is : " + list);
                }
                else{
                        System.out.println("Error : No such index to remove");
                }
        }
}
```

Genie Ashwani

Program to check whether the list contains element or not: contains() method returns true if the list has specified element.

```
import java.util.*;
class Code
{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
		List<Integer> list = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++)
			list.add(i);
		System.out.println("List is : " + list);
		System.out.print("Enter element to check in list : ");
		int ele = sc.nextInt();
		if(list.contains(ele))
			System.out.println("Yes element is present in list");
		else
			System.out.println("No such element in list");
	}
}
```

Program display the index value of element: indexOf() method returns index of specified element. It returns -1 if no such element in the list.

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		List<Integer> list = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++)
			list.add(i);
		System.out.println("List is : " + list);
		System.out.print("Enter element to find index value : ");
		int ele = sc.nextInt();
		int index = list.indexOf(ele);
		if(index!=-1)
			System.out.println("Index value is : " + index);
		else
			System.out.println("No such element in list");
	}
}
```

Program to replace the existing value: set(int index, E e) method replace the index element with specified element.

Instructions to code:

- Create ArrayList with elements.
- Read the element to replace
- Check the element is present or not in the list using contains() method.
- If the element is present,
    - Read the new element to replace with.
- If the element is not present,
    - Display error message.

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		List<Integer> list = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++){
			list.add(i);
		}
		System.out.println("List is : " + list);
		System.out.print("Enter element to replace : ");
		int x = sc.nextInt();
		if(list.contains(x)) {
			System.out.print("Enter new element : ");
			int y = sc.nextInt();

			int loc = list.indexOf(x);
			list.set(loc, y);
			System.out.println("Updated list : " + list);
		}
		else
			System.out.println("No such element in list");
	}
}
```

For-each loop:

- It is also called enhanced for loop.
- It is since JDK5
- For-each loop provides easy syntax to process elements of Array or Collection.

Limitations:

- For-each loop can process elements only in forward direction.
- For-each loop can process elements one by one only.

Syntax:

```
for (datatype var : Array/Collection ) {
        statements ;
}
```

Program to display ArrayList using for-each loop:

```
import java.util.*;
class Code {
        public static void main(String[] args) {
                List<Integer> list = new ArrayList<Integer>();
                for(int i=1 ; i<=5 ; i++)
                        list.add(i*5);

                System.out.println("List is :
                ");for(Integer x : list)
                        System.out.println(x);
        }
}
```

Display ArrayList element by element using for-loop: get(int index) method is used to retrieve each element using its index.

```
import java.util.*;
class Code {
        public static void main(String[] args) {
                List<Integer> list = new ArrayList<Integer>();
                for(int i=1 ; i<=5 ; i++)
                        list.add(i*5);

                System.out.println("List is : ");
                for(int i=0 ; i<=list.size()-1 ; i++)
                        System.out.println(list.get(i));
        }
}
```

Genie Ashwani

Program to display ArrayList in Reverse Order:

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		List<Integer> list = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++)
			list.add(i*5);

		System.out.println("List is : ");
		for(int i=list.size()-1 ; i>=0 ; i--)
			System.out.println(list.get(i));
	}
}
```

Program to Merge 2 ArrayLists: addAll(Collection c) method is used to merge 2 lists.

```
import java.util.*;
class Code
{
	public static void main(String[] args) {
		List<Integer> a1 = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++)
			a1.add(i*5);
		System.out.println("a1 list is : " + a1);

		List<Integer> a2 = new ArrayList<Integer>();
		for(int i=5 ; i>=1 ; i--)
			a2.add(i*5);
		System.out.println("a1 list is : " + a1);

		a1.addAll(a2);
		System.out.println("a1 list after merge : " + a1);
	}
}
```

Genie Ashwani

Iterator:

- It is an interface.
- Iterator providing methods to iterator any collection.
- iterator() method returns Iterator object of any collection.

Methods:

1. **boolean hasNext():** checks the next element is present or not to iterate.
2. **Object next():** returns the next element of iterator object.

Program to display ArrayList using Iterator:

```
import java.util.*;
class Code
{
	public static void main(String[] args)
	{
		List<Integer> list = new ArrayList<Integer>();
		for(int i=1 ; i<=5 ; i++)
			list.add(i*5);

		System.out.println("Display using Iterator :");
		Iterator<Integer> itr = list.iterator();
		while(itr.hasNext())
		{
			Integer ele = itr.next();
			System.out.println(ele);
		}
	}
}
```

When we use for/for-each/iterator?

For-loop	For-each loop	Iterator
Index based.	Not index based.	Not index based.
Process only List(index based)	Process List, Set and Map	Process List, Set and Map
Use get(index) method	Do not use any other method	Do not use any other method

ListIterator:

- It is an interface
- listIterator() method returns ListIterator object.
- Using ListIterator, we can iterate elements,
    - In Forward direction
    - In Backward direction
    - From specified index value

Iterator List in Forward Direction using hasNext() and next() methods:

```
List<Integer> list = new ArrayList<Integer>();
for(int i=1 ; i<=5 ; i++){
        list.add(i*5);
}
ListIterator<Integer> itr =
list.listIterator();while(itr.hasNext())
{
        System.out.println(itr.next());
}
```

Iterator List in Backward Direction using hasPrevious() and previous() methods:

```
List<Integer> list = new ArrayList<Integer>();
for(int i=1 ; i<=5 ; i++){
        list.add(i*5);
}
ListIterator<Integer> itr =
list.listIterator(list.size());while(itr.hasPrevious())
{
        System.out.println(itr.previous());
}
```

Display list from specified index value:

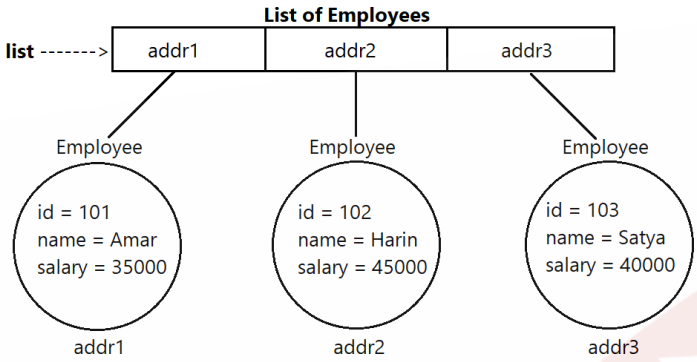
```
List<Integer> list = new ArrayList<Integer>();
for(int i=1 ; i<=10 ; i++){
        list.add(i*5);
}
ListIterator<Integer> itr =
list.listIterator(5);while(itr.hasNext())
{
        System.out.println(itr.next());
}
```

Genie Ashwani

List of Employee Objects
(Parameterized constructor approach)

List of Objects:

- Collections are mainly used to store and process information of Employees, Students, Customers, Products, Books, Accounts etc.
- Object is a set of dissimilar elements. For example, Employee has ID, Name and Salary.
- We create objects with details and store the object into collection as follows.



Program to create and display List of Employees:

1. **Employee.java:** contains Employee class
2. **Main.java:** contains code of creating ArrayList with Employees and display.

Approach1: (Create 3 employee objects directly and add to list)
Employee.java:

- Create Employee class with instance variables id, name, salary
- Define parameterized constructor to initialize the object.

```
class Employee
{
        int id;
        String name;
        double salary;
        Employee(int id, String name, double salary) {
                this.id = id;
                this.name = name;
                this.salary = salary;
        }
}
```

Main.java:

- Create 3 Employee objects and add to List
- Display details using for-each loop

```
import java.util.*;
class Main {
	public static void main(String[] args) {
		List<Employee> list = new ArrayList<Employee>();

		Employee e1 = new Employee(101, "Amar", 35000);
		Employee e2 = new Employee(102, "Harin", 45000);
		Employee e3 = new Employee(103, "Satya", 40000);
		list.add(e1);
		list.add(e2);
		list.add(e3);
		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.id + " , " + e.name + " , " + e.salary);
		}
	}
}
```

You can directly add objects to the list as follows:

```
import java.util.*;
class Main {
	public static void main(String[] args) {
		List<Employee> list = new ArrayList<Employee>();

		list.add(new Employee(101, "Amar",
		35000)); list.add(new Employee(102,
		"Harin", 45000));list.add(new Employee(103,
		"Satya", 40000));

		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.id + " , " + e.name + " , " + e.salary);
		}
	}
}
```

Display using for loop:

```
System.out.println("Details are : ");
for(int i=0 ; i<=list.size()-1 ; i++)
{
        Employee e = list.get(i);
        System.out.println(e.id + " , " + e.name + " , " + e.salary);
}
```

Display Employees List in reverse order:

- We must use for() loop to iterate in reverse order.
- For-each loop can move only in forward direction.

```
System.out.println("Details are : ");
for(int i=list.size()-1 ; i>=0 ; i--)
{
        Employee e = list.get(i);
        System.out.println(e.id + " , " + e.name + " , " + e.salary);
}
```

Display using Iterator:

```
System.out.println("Details are : ");
Iterator<Employee> itr = list.iterator();
while(itr.hasNext())
{
        Employee e = itr.next();
        System.out.println(e.id + " , " + e.name + " , " + e.salary);
}
```

Display reverse list using ListIterator:

```
System.out.println("Details are : ");
ListIterator<Employee> itr =
list.listIterator(list.size());while(itr.hasPrevious())
{
        Employee e = itr.previous();
        System.out.println(e.id + " , " + e.name + " , " + e.salary);
}
```

Genie Ashwani

Approach-2: (Create Employee objects by collecting details from arrays)

Employee.java:

- Create Employee class with instance variables id, name, salary
- Define parameterized constructor to initialize the object.

```
class Employee
{
	int id;
	String name;
	double salary;
	Employee(int id, String name, double salary) {
		this.id = id;
		this.name = name;
		this.salary = salary;
	}
}
```

Main.java:

- Collect values from Arrays to create Employee objects.
- Display details using for-each loop

```
import java.util.*;
class Main
{
	public static void main(String[] args) {
		int[] ids = {101, 102, 103, 104, 105};
		String[] names = {"Amar", "Annie", "Harini", "Satya", "Jai"};
		double[] salaries = {23000, 56000, 43000, 48000, 16000};

		List<Employee> list = new ArrayList<Employee>();
		for (int i=0 ; i<=ids.length-1 ; i++){
			Employee e = new Employee(ids[i], names[i], salaries[i]);
			list.add(e);
		}

		System.out.println("Details are : ");
		for(Employee e : list){
			System.out.println(e.id + " , " + e.name + " , " + e.salary);
		}
	}
}
```

Approach-3: (Read details using Scanner class)
Employee.java:

```
class Employee
{
	int id;
	String name;
	double salary;
	Employee(int id, String name, double salary) {
		this.id = id;
		this.name = name;
		this.salary = salary;
	}
}
```

Main.java: Create List with 5 Employee details by reading through Scanner.

```
import java.util.*;
class Main
{
	public static void main(String[] args)
	{
		List<Employee> list = new ArrayList<Employee>();
		Scanner sc = new Scanner(System.in);

		System.out.println("Enter 5 Employee details : ");
		for (int i=1 ; i<=5 ; i++)
		{
			System.out.println("Enter Emp-" + i + " details : ");
			int id = sc.nextInt();
			String name = sc.next();
			double salary = sc.nextDouble();
			Employee e = new Employee(id, name, salary);
			list.add(e);
		}

		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.id + " , " + e.name + " , " + e.salary);
		}
	}
}
```

Genie Ashwani

Approach-4: (Store objects to list until user quits)
Employee.java:

- Create Employee class with instance variables id, name, salary
- Define parameterized constructor to initialize the object.

Main.java: Read and Add employee details until end user quits.

```
import java.util.*;
class Main
{
	public static void main(String[] args)
	{
		List<Employee> list = new ArrayList<Employee>();
		Scanner sc = new Scanner(System.in);

		while(true)
		{
			System.out.println("Enter Emp details to add : ");
			int id = sc.nextInt();
			String name = sc.next();
			double salary = sc.nextDouble();
			Employee e = new Employee(id, name, salary);
			list.add(e);

			System.out.print("Do you want to add another record(yes/no) : ");
			String choice = sc.next();
			if(choice.equals("no"))
			{
				break;
			}
		}

		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.id + " , " + e.name + " , " + e.salary);
		}
	}
}
```

Approach-5: (Using BufferedReader class)
Employee.java:

- Create Employee class with instance variables id, name, salary
- Define parameterized constructor to initialize the object.

Main.java: Read and Add employee details until end user quits using **BufferedReader**.

```
import java.util.*;
import java.io.*;
class Main
{
	public static void main(String[] args) throws Exception {
		List<Employee> list = new ArrayList<Employee>();
		BufferedReader br = null;
		try{
			br = new BufferedReader(new InputStreamReader(System.in));
			while(true)
			{
				System.out.println("Enter Emp details to add : ");
				int id = Integer.parseInt(br.readLine());
				String name = br.readLine();
				double salary = Double.parseDouble(br.readLine());
				Employee e = new Employee(id, name, salary);
				list.add(e);

				System.out.print("Do you add another record(yes/no) : ");
				String choice = br.readLine();
				if(choice.equals("no"))
					{break;
				}
			}

			System.out.println("Details are : ");
			for(Employee e : list) {
				System.out.println(e.id + " , " + e.name + " , " + e.salary);
			}
		}
		finally{
			if(br!=null)
				br.close();
		}
	}
}
```

List of Employee Objects
(POJO class approach)

POJO class: (Plain Old Java Object)

- POJO rules are:
  - Class is public
  - Variables are private
  - Every variable has get() and set() methods.

Approach1: (Construct objects from Arrays)

Employee.java: Create Employee POJO class

```
public class Employee
{
	private int id;
	private String name;
	private double salary;
	public void setId(int id) {
		this.id = id;
	}
	public void setName(String name) {
		this.name = name;
	}
	public void setSalary(double salary) {
		this.salary = salary;
	}
	public int getId() {
		return this.id;
	}
	public String getName() {
		return this.name;
	}
	public double getSalary() {
		return this.salary;
	}
}
```

Genie Ashwani

Main.java:

Construct objects by reading using Scanner)

```
import java.util.*;
class Main
{
	public static void main(String[] args) {
		List<Employee> list = new ArrayList<Employee>();
		Scanner sc = new Scanner(System.in);

		while(true)
		{
			System.out.println("Enter Emp details : ");
			int id = sc.nextInt();
			String name = sc.next();
			double salary = sc.nextDouble();

			Employee e = new Employee();
			e.setId(id);
			e.setName(name);
			e.setSalary(salary)
			;

			list.add(e);

			System.out.print("Want to add one more(y/n) :");
			if(sc.next().charAt(0) == 'n')
			{
				break;
			}
		}

		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary());
		}
	}
}
```

ArrayList – Case Studies

Write code for following instructions:

- Define Employee POJO with variables id, name, salary, dept, location.
- Create an ArrayList of Employee type and store following values from arrays.

Id	Name	Salary	Dept	Location
101	Amar	30000	20	Hyderabad
102	Hareen	35000	10	Chennai
103	Sathya	40000	20	Bangalore
104	Annie	45000	20	Hyderabad
105	Raji	42000	30	Pune
106	Harsha	50000	10	Bangalore

Employee.class:

```
class Employee {
	private int id;
	private String name;
	private double salary;
	private int dept;
	private String location;
	int getId(){
		return this.id;
	}
	String getName(){
		return this.name;
	}
	double getSalary(){
		return this.salary;
	}
	int getDept(){
		return this.dept;
	}
	String getLocation(){
		return this.location;
	}
	void setId(int id){
		this.id = id;
	}
	void setName(String name){
		this.name = name;
	}
	void setSalary(double salary){
		this.salary = salary;
	}
```

Genie Ashwani

```
	void setDept(int dept){
		this.dept = dept;
	}
	void setLocation(String location){
		this.location = location;
	}
}
```

Main.java:

```
import java.util.*;
class Main {
	public static void main(String[] args)  {
		int[] ids = {101, 102, 103, 104, 105, 106};
		String[] names = {"Amar", "Hareen", "Sathya", "Annie", "Raji", "Harsha"};
		double[] salaries = {30000, 35000, 40000, 45000, 42000, 50000};
		int[] depts = {20, 10, 20, 20, 30, 10};
		String[] locations = {"Hyderabad", "Chennai", "Bangalore", "Hyderabad",
"Pune", "Bangalore"};

		List<Employee> list = new ArrayList<Employee>();
		for (int i=0 ; i<=ids.length-1 ; i++) {
			Employee e = new Employee();
			e.setId(ids[i]);
			e.setName(names[i]);
			e.setSalary(salaries[i]);
			e.setDept(depts[i]);
			e.setLocation(locations[i]);
			list.add(e);
		}
		System.out.println("Details are : ");
		for(Employee e : list)
		{
			System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() +" , "
+ e.getDept() + " , " + e.getLocation());
		}
	}
}
```

Display details using for loop:

```
for(int i=0 ; i<=list.size()-1 ; i++)
{
	Employee e = list.get(i);
	System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +e.getDept() + "
, " + e.getLocation());
}
```

Display details in reverse order:

```
for(int i=list.size()-1 ; i>=0 ; i--)
{
	Employee e = list.get(i);
	System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +e.getDept() + "
, " + e.getLocation());
}
```

Display Employee details whose ID is 103:

```
boolean found=false;
for(Employee e : list)
{
	if(e.getId() == 103)
	{
		System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
		found = true;
		break;
	}
}
if(!found)
{
	System.out.println("ID 103 doesn't exist");
}
```

Display Employee details belongs to Hyderabad:

```
int count=0;
for(Employee e : list)
{
```

```
        if(e.getLocation().equals("Hyderabad"))
        {
                System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
        }
}
if(count==0)
{
        System.out.println("No employee belongs to Hyderabad");
}
```

Display Employee details belongs in department 20 or 30:

```
int count=0;
for(Employee e : list)
{
        if(e.getDept()==20 || e.getDept()==30)
        {
                System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
        }
}
if(count==0)
{
        System.out.println("No employee belongs depts 20 or 30");
}
```

Display employee details those who not belongs to Hyderabad.

```
int count=0;
for(Employee e : list)
{
        if(!(e.getLocation().equals("Hyderabad")))
        {
                System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
```

Genie Ashwani

```
        }
}
if(count==0)
{
        System.out.println("No employee records founds");
}
```

Display details belongs to department 20 and not belongs to Hyderabad:

```
int count=0;
for(Employee e : list)
{
        if(e.getDept()==20 && !(e.getLocation().equals("Hyderabad")))
        {
                System.out.println(e.getId() + " ," + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
        }
}
if(count==0)
{
        System.out.println("No employee records founds");
}
```

Count how many employees working in both Hyderabad and Bangalore locations:

```
int count=0;
for(Employee e : list)
{
        String loc = e.getLocation();
        if(loc.equals("Hyderabad") || loc.equals("Bangalore"))
        {
                count++;
        }
}
System.out.println("Count is : " + count);
```

Genie Ashwani

Check the Employee with name "Amar" present or not:

```
boolean found=false;
for(Employee e : list)
{
	if(e.getName().equals("Amar"))
	{
		System.out.println("Found with ID : " + e.getId());
		found=true;
		break;
	}
}
if(!found)
{
	System.out.println("Amar not present");
}
```

Display details whose salary greater than 35000:

```
int count=0;
for(Employee e : list)
{
	if(e.getSalary()>35000)
	{
		System.out.println(e.getId() + " , " + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
		count++;
	}
}
if(count==0)
{
	System.out.println("No employee found");
}
```

Display details whose salary between 30000 and 40000:

```
int count=0;
for(Employee e : list)
{
	if(e.getSalary()>30000 && e.getSalary()<40000)
```

Genie Ashwani

```
        {
                System.out.println(e.getId() + " , " + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
        }
}
if(count==0)
{
        System.out.println("No employee found");
}
```

Display details whose salary below 40000 and not belongs to Hyderabad:

```
int count=0;
for(Employee e : list)
{
        if(e.getSalary()<40000 && !(e.getLocation().equals("Hyderabad")))
        {
                System.out.println(e.getId() + " , " + e.getName() + " , " + e.getSalary() + " , " +
e.getDept() + " , " + e.getLocation());
                count++;
        }
}
if(count==0)
{
        System.out.println("No employee found");
}
```

Genie Ashwani

ArrayList Operations – Menu Driven Approach

Following program explains how to perform ArrayList operations such as Append, Insert,Replace, Update, Remove, Sort, Reverse and Display:

```
import java.util.*;
class Main
{
	public static void main(String[] args)
	{
		List<Integer> list = new ArrayList<Integer>();
		Scanner sc = new Scanner(System.in);
		while(true)
		{
			System.out.println("1.Append");
			System.out.println("2.Insert");
			System.out.println("3.Replace");
			System.out.println("4.Remove");
			System.out.println("5.Display");
			System.out.println("6.Sort");
			System.out.println("7.Reverse");
			System.out.println("8.Quit");

			System.out.print("Enter choice : ");
			int ch = sc.nextInt();
			if(ch==1)
			{
				System.out.print("Enter element to append : ");
				int ele = sc.nextInt();
				list.add(ele);
				System.out.println("Element added");
			}
			else if(ch==2)
			{
				System.out.print("Enter index : ");
				int index = sc.nextInt();

				if(index>=0 && index<=list.size()-1){
```

Genie Ashwani

```
                        System.out.print("Enter element : ");
                        int ele = sc.nextInt();
                        list.add(index, ele);
                        System.out.println("Element inserted");
                }
                else
                        System.out.println("No such location");
        }
        else if(ch==3)
        {
                System.out.print("Enter element to replace : ");
                int ele = sc.nextInt();

                if(list.contains(ele)){
                        int index = list.indexOf(ele);
                        System.out.print("Enter new element : ");
                        int x = sc.nextInt();
                        list.set(index, x);
                        System.out.println("Element replaced");
                }
                else
                        System.out.println("No such element in list");
        }
        else if(ch==4)
        {
                System.out.print("Enter element to remove : ");
                int ele = sc.nextInt();
                if(list.contains(ele)){
                        int index = list.indexOf(ele);
                        list.remove(index);
                        System.out.println("Element removed");
                }
                else
                        System.out.println("No such element to remove");
        }
        else if(ch==5)
        {
```

Genie Ashwani

```
                        if(list.isEmpty())
                                System.out.println("Empty list");
                        else
                                System.out.println("List is : " + list);
                }
                else if(ch==6)
                {
                        Collections.sort(list);
                        System.out.println("List
                        sorted");
                }
                else if(ch==7)
                {
                        Collections.reverse(list);
                        System.out.println("List
                        reversed");
                }
                else if(ch==8)
                {
                        System.out.println("End");
                        System.exit(1);
                }
                else
                        System.out.println("Invalid choice");
        }
    }
}
```

ArrayList – Employee CRUD – Menu Driven Approach

This program explains how to add employee details, display details of specific ID, remove employee and update the details of employee:

```
import java.util.*;
class Main
{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        List<Employee> list = new ArrayList<Employee>();
        while(true){
```

```
System.out.println("1.Add Record");
System.out.println("2.Display Record");
System.out.println("3.Display All");
System.out.println("4.Update Record");
System.out.println("5.Delete Record");
System.out.println("6.Exit");

System.out.print("Enter choice : ");
int ch = sc.nextInt();

if(ch==1){
        System.out.println("Enter details
        :");int id = sc.nextInt();
        String name = sc.next();
        double salary = sc.nextDouble();

        Employee e=new Employee(id,name,salary);
        list.add(e);
        System.out.println("Record Added");
}
else if(ch==2){
       if(list.isEmpty()){
              System.out.println("empty list");
        }
        else
        {     System.out.print("Enter id : ");
              int id = sc.nextInt();

               boolean found=false;
              for(Employee e : list){
                    if(e.id == id){
                          System.out.println("Name : " + e.name);
                          System.out.println("Salary : " + e.salary);
                          found = true;
                          break;
                    }
              }
```

Genie Ashwani

```
                    if(!found)
                        System.out.println("Invalid ID");
                }
            }
            else if(ch==3){
                if(list.isEmpty()){
                    System.out.println("Empty list");
                }
                else
                {   System.out.println("Details : ");
                    for(Employee e : list){
                        System.out.println("Name : " + e.name);
                        System.out.println("Salary : " + e.salary);
                    }
                }
            }
            else if(ch==4){
                if(list.isEmpty()){
                    System.out.println("Empty list");
                }
                else
                {   System.out.print("Enter id : ");
                    int id = sc.nextInt();

                    boolean found=false;
                    for(Employee e : list){
                        if(e.id == id){
                            System.out.print("Enter sal to update: ");
                            double salary = sc.nextDouble(); e.salary
                            = salary; System.out.println("Record
                            updated"); found = true;
                            break;
                        }
                    }
                    if(!found)
                        System.out.println("Invalid ID");
```

Genie Ashwani

```
                    }
                }
                else if(ch==5){
                    if(list.isEmpty()){
                        System.out.println("Empty list");
                    }
                    else
                    {
                        System.out.print("Enter id : ");
                        int id = sc.nextInt();

                        boolean found=false;
                        for(Employee e : list)
                        {
                            if(e.id == id){
                                int index = list.indexOf(e);
                                list.remove(index);
                                System.out.println("Removed");
                                found = true;
                                break;
                            }
                        }
                        if(!found)
                            System.out.println("Invalid ID");
                    }
                }
                else if(ch==6){
                    System.out.println("End");
                    System.exit(1);
                }
                else
                    System.out.println("Invalid choice");
            }
        }
}
```

Genie Ashwani

Vector Collection

Vector:

- Vector implements List.
- Vector allow duplicates and follow insertion order.
- Vector is synchronized by default.

```
import java.util.*;
class Code {
	public static void main(String[] args)
	{
		Vector<Integer> v = new Vector<Integer>();
		for (int i=1 ; i<=5 ; i++){
			v.add(i*5);
		}
		System.out.println("Vector : " + v);
	}
}
```

Enumeration:

- Vector is legacy(old) class since first version of JDK.
- Enumeration interface used to process vector element by element.
- elements() method of Vector class returns Enumeration-interface.

Methods of Enumeration:

1. hasMoreElements(): is used to check the element is present or not in Enumeration
2. nextElement(): returns the next element in the enumeration.

```
import java.util.*;
class Code {
	public static void main(String[] args)
	{
		Vector<Integer> v = new Vector<Integer>();
		for (int i=1 ; i<=5 ; i++){
			v.add(i*5);
		}
		System.out.println("Vector : ");
		Enumeration<Integer> en = v.elements();
		while(en.hasMoreElements())
		{
			Integer x = en.nextElement();
			System.out.println(x);
		}
	}
   }
```

Genie Ashwani

ArrayList is not synchronized: We get odd results when we try to add elements into ArrayList from multiple threads.

```
import java.util.*;
class Test
{
	static ArrayList<Integer> list = new ArrayList<Integer>();
}
class First extends Thread
{
	public void run(){
		for (int i=1 ; i<=100000 ; i++)
		{
			Test.list.add(i);
		}
	}
}
class Second extends Thread
{
	public void run(){
		for (int i=1 ; i<=100000 ; i++)
		{
			Test.list.add(i);
		}
	}
}
class Code
{
	public static void main(String[] args) throws Exception {
		First f = new First();
		Second s = new Second();
		f.start();
		s.start();
		f.join();
		s.join();
		System.out.println("List size is : " + Test.list.size());
	}
}
```

Output: **List size is : 166987**

Vector is synchronized by default: Vector is thread safe, hence we get perfect results when wetry to add elements from multiple threads

```
import java.util.*;
class Test
{
	static Vector<Integer> list = new Vector<Integer>();
}
class First extends Thread
{
	public void run(){
		for (int i=1 ; i<=100000 ; i++)
		{
			Test.list.add(i);
		}
	}
}
class Second extends Thread
{
	public void run(){
		for (int i=1 ; i<=100000 ; i++)
		{
			Test.list.add(i);
		}
	}
}
class Code
{
	public static void main(String[] args) throws Exception
	{
		First f = new First();
		Second s = new Second();
		f.start();
		s.start();
		f.join();
		s.join();
		System.out.println("Vector size is : " + Test.list.size());
	}
}
```

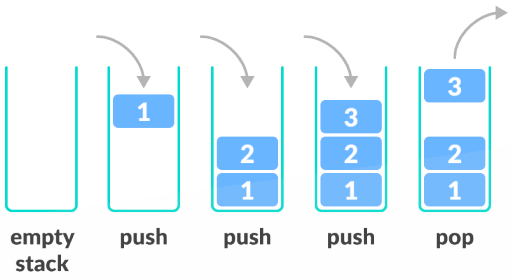
Output: **Vector size is : 200000**

Genie Ashwani

Stack - Collection

Stack:

- Stack is an extension of Vector class.
- It follows LIFO – Last In First Out Rule



Methods are:

1. boolean empty() : Tests the stack is empty or not
2. Object peek(): returns the top element of stack but not remove
3. Object pop(): returns the top element of stack and removes
4. void push(Object e): push element on to the stack

```
import java.util.*;
class Code
{
	public static void main(String[] args) throws Exception {
		Stack<Integer> stk = new Stack<Integer>();
		stk.push(10);
		stk.push(20);
		stk.push(30);
		stk.push(40);
		System.out.println("Stack is : " + stk);

		System.out.println("Pop : " + stk.pop());
		System.out.println("Pop : " + stk.pop());
		System.out.println("Stack is : " + stk);

		stk.push(50);
		stk.push(60);
		System.out.println("Stack is : " + stk);

		System.out.println("Peek : " + stk.peek());
		System.out.println("Peek : " + stk.peek());
		System.out.println("Stack is : " + stk);
	}
}
```

Genie Ashwani

Stack Operations – Menu Driven Program

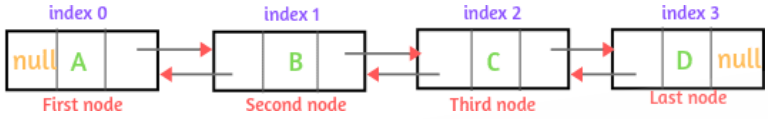
```
import java.util.*;
class Code {
	public static void main(String[] args) throws Exception {
		Scanner sc = new Scanner(System.in);
		Stack<Integer> stk = new Stack<Integer>();
		while(true){
			System.out.println("1.Push \n2.Pop \n3.Display \n4.Peek
			\n5.Quit");System.out.print("Enter choice : ");
			int ch = sc.nextInt();
			if(ch==1){
				System.out.print("Enter element to push : ");
				int ele = sc.nextInt();
				stk.push(ele);
				System.out.println("Element Pushed");
			}
			else if(ch==2){
				if(stk.empty())
					System.out.println("Empty stack");
				else
					System.out.println("Pop : " + stk.pop());
			}
			else if(ch==3){
				if(stk.empty())
					System.out.println("Empty stack");
				else
					System.out.println("Stack is : " + stk);
			}
			else if(ch==4){
				if(stk.empty())
					System.out.println("Empty stack");
				else
					System.out.println("Peek : " + stk.peek());
			}
			else if(ch==5){
				System.exit(1);
			}
			else
				System.out.println("Invalid choice");
		}
	}
}
```

Genie Ashwani

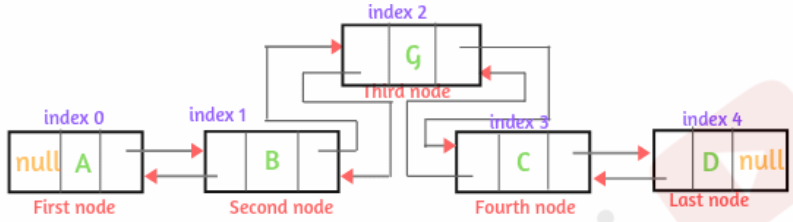
Linked List

LinkedList:

- LinkedList implements List
- LinkedList allow duplicates and ordered collection.
- Linked List store elements in the form of nodes and connect with links.



- Accessing elements – slower in LinkedList
- Insertions and Deletions are faster – no shifting of elements.



You can see that one node is created with element G and simply changes the next and previous pointer only. No shift of operation has occurred.

Methods are:

boolean add(E e)	Appends the specified element to the end of this list.
void add(int index, E element)	Inserts the specified element at the specified position
void addFirst(E e)	Inserts the specified element at the beginning of this list.
void addLast(E e)	Appends the specified element to the end of this list.
void clear()	Removes all of the elements from this list.
boolean contains(Object o)	Returns true if this list contains the specified element.
E get(int index)	Returns the element at the specified position in this list.
E getFirst()	Returns the first element in this list.
E getLast()	Returns the last element in this list.
Iterator descendingIterator()	Returns an iterator over the elements in this reverse.
int indexOf(Object o)	Returns element index or else -1
ListIterator listIterator(int index)	Create iterator from specified index.
E remove(int index)	Removes the element at the specified position in this list.
E removeFirst()	Removes and returns the first element from this list.
E removeLast()	Removes and returns the last element from this list.
E set(int index, E element)	Replace index element with specified element.
int size()	Returns the number of elements in this list.

Genie Ashwani

```
import
java.util.LinkedList;class
Demo{
	public static void main(String[] args){
		LinkedList<String> linkedList = new LinkedList<>();
		linkedList.add("Apple");
		linkedList.add("Banana");
		linkedList.add("Orange");
		linkedList.addFirst("Mango");
		linkedList.addLast("Grapes");
		System.out.println("LinkedList: " + linkedList);

		// Retrieving
		String first = linkedList.getFirst();
		String last = linkedList.getLast();
		System.out.println("First: " +
		first);System.out.println("Last: " +
		last);

		String element = linkedList.get(2);
		System.out.println("Element at position 2: " + element);

		// Searching
		boolean containsBanana = linkedList.contains("Banana");
		System.out.println("Contains Banana: " + containsBanana);

		linkedList.removeFirst(
		);
		linkedList.removeLast()
		;
		System.out.println("Removing first and last elements: " + linkedList);

		linkedList.remove(1);
		System.out.println("LinkedList after removing : " + linkedList);

		boolean isEmpty = linkedList.isEmpty();
		System.out.println("Is Empty: " + isEmpty);

		int size = linkedList.size();
		System.out.println("Size: " + size);

		linkedList.clear();
		System.out.println("LinkedList after clearing all elements: " + linkedList);
	}
}
```

Set Interface

Set:

- Set doesn't allow duplicates.
- Set is not index based.

HashSet Methods are:

boolean add(E e)	Adds the specified element to this set if it is not already present.
void clear()	Removes all of the elements from this set.
Object clone()	Returns a shallow copy: the elements themselves are not cloned.
boolean contains(Object o)	Returns true if this set contains the specified element.
boolean isEmpty()	Returns true if this set contains no elements.
Iterator<E> iterator()	Returns an iterator over the elements in this set.
boolean remove(Object o)	Removes the specified element from this set if it is present.
int size()	Returns the number of elements in this set (its cardinality).

Note: Set is not providing any methods to perform index-based operations.

Implementations are:

1. **HashSet:** It doesn't maintain insertion order
2. **LinkedHashSet:** It maintains insertion order of elements.
3. **TreeSet:** It maintains sorted order of elements.

Program to store elements into HashSet and display:

```
import java.util.*;
class Code {
	public static void main(String[] args) {
		Set<Integer> set = new HashSet<Integer>();
		set.add(50);
		set.add(40);
		set.add(30);
		set.add(20);
		set.add(10);
		System.out.println("Set : " + set);
	}
}
```

Output: Random order of elements

Program to store elements into LinkedHashSet and display:

```
import java.util.*;
class Code {
	public static void main(String[] args) {
		Set<Integer> set = new LinkedHashSet<Integer>();
		set.add(50);
		set.add(40);
```

```
                set.add(30);
                set.add(20);
                set.add(10);
                System.out.println("Set : " + set);
        }
}
```
Output: 50, 40, 30, 20, 10

Program to store elements into TreeSet and display:

```
import java.util.*;
class Code {
        public static void main(String[] args) {
                Set<Integer> set = new TreeSet<Integer>();
                set.add(50);
                set.add(40);
                set.add(30);
                set.add(20);
                set.add(10);
                System.out.println("Set : " + set);
        }
}
```
Output: 10, 20, 30, 40, 50

Remove duplicates in ArrayList:

- ArrayList is ordered and allow duplicates.
- To remove duplicates in array, we simply convert into Set and display

```
import java.util.*;
class Code {
        public static void main(String[] args) {
                List<Integer> list = new ArrayList<Integer>();
                for(int i=1 ; i<=4 ; i++) {
                        list.add(i+2)
                        ;
                        list.add(i+3)
                        ;
                }
                System.out.println("List : " + list);
                Set<Integer> set = new HashSet<Integer>(list);
                System.out.println("Set : " + set);
        }
}
```

Output: List : [3, 4, 4, 5, 5, 6, 6, 7]
Set : [3, 4, 5, 6, 7]

Map Interface

Map:

- Store values using keys
  Map = {key=value, key=value, key=value ….}
- Keys must be unique in Map
- Values can be duplicated.
- For example, book names(unique) with prices(duplicates)
  - Books = {C=300.0 , C++=300.0, Java=350.0, Python=330.0};

Methods are:

Method	Description
put(K key, V value)	store value using key.
V get(Object key)	return value of specified key.
boolean isEmpty()	Returns true if this map contains key-values.
void clear()	Removes all elements.
boolean containsKey(Object key)	Returns true if map contains specified key.
Set<K> keySet()	Returns a Set of keys contained in this map.
remove(Object key)	Removes key-value of specified key.
replace(K key, V value)	Replace the value of specified key with given value.
int size()	Returns the number of key-values in map.
Collection<V> values()	Returns Collection of values in this map.

Program to create HashMap and display:

```
import java.util.*;
class Code {
	public static void main(String[] args)
	{
		Map<Integer,String> map = new HashMap<Integer,String>();
		map.put(10, "Ten");
		map.put(20, "Twenty");
		map.put(30, "Thirty");
		map.put(40, "Fourty");
		map.put(50, "Fifty");
		System.out.println("Map : " + map);
	}
}
```

Implementations of Map:

1. **HashMap:** doesn't maintain insertion order. Allows only one null key
2. **LinkedHashMap:** maintains insertion order. Allows only one null key
3. **TreeMap:** maintains sorted order of keys. It doesn't allow null key.
4. **Hashtable:** It is call legacy class. It maintains sorted order. It doesn't allow null key.

```
import java.util.*;
class Code
{
	public static void main(String[] args)
	{
		//Map<Integer,String> map = new LinkedHashMap<Integer,String>();
		Map<Integer,String> map = new TreeMap<Integer,String>();
		map.put(50, "Fifty");
		map.put(40, "Fourty");
		map.put(30, "Thirty");
		map.put(20, "Twenty");
		map.put(10, "Ten");
		System.out.println("Map : " + map);
	}
}
```

Set<K> keySet():

- We cannot iterate the map object either by using Iterator or using for-each loop.
- First we need to collect all keys of map using keySet() method.
- We iterate keys set and get values by specifying each key.

```
import java.util.*;
class Code
{
	public static void main(String[] args) throws Exception
	{
		Map<Integer, String> map = new HashMap<Integer, String>();
		map.put(10, "Ten");
		map.put(20, "Twenty");
		map.put(30, "Thirty");
		map.put(40, "Fourty");

		System.out.println("Map is : ");
		Set<Integer> keys = map.keySet();
		for(Integer key : keys)
		{
```

```
                String value = map.get(key);
                System.out.println(key + " = " + value);
            }
        }
}
```

Iterate Map through Iterator:

- Collect the keys using keySet() method.
- Create Iterator from the keys Set.
- Iterate the object and get Values by specifying keys.

```
import java.util.*;
class Code
{
    public static void main(String[] args) throws Exception
    {
        String[] books = {"C", "C++", "Java", "Python", "Android"};
        double[] prices = {200.0, 300.0, 250.0, 200.0, 250.0};

        Map<String, Double> map = new HashMap<String, Double>();
        for(int i=0 ; i<=books.length-1 ; i++)
        {
            map.put(books[i], prices[i]);
        }

        System.out.println("Map is : ");
        Set<String> keys = map.keySet();
        Iterator<String> itr = keys.iterator();
        while(itr.hasNext())
        {
            String key = itr.next();
            Double value = map.get(key);
            System.out.println(key + " = " + value);
        }
    }
}
```

Menu Driven Program (Books and Prices)

```
import java.util.*;
class Main
{
    public static void main(String[] args)
    {
```

```
Scanner sc = new Scanner(System.in);
Map<String, Double> map = new HashMap<String, Double>();
while(true)
{
	System.out.println("1. Add Book");
	System.out.println("2. Update Book");
	System.out.println("3. Display Book");
	System.out.println("4. Remove Book");
	System.out.println("5. Quit");

	System.out.print("Enter your choice : ");
	int ch = sc.nextInt();

	if(ch==1)
	{
		System.out.print("Enter Book Name : ");
		String name = sc.next();
		if(map.containsKey(name))
		{
			System.out.println("Book already exists");
		}
		else
		{
			System.out.print("Enter Price : ");
			double price = sc.nextDouble();
			map.put(name, price);
			System.out.println("Book added");
		}
	}
	else if(ch==2)
	{
		System.out.print("Enter Book Name : ");
		String name = sc.next();
		if(map.containsKey(name))
		{
			System.out.print("Enter Price : ");
			double price = sc.nextDouble();
			map.replace(name, price);
			System.out.println("Book updated");
		}
		else
			System.out.println("Error : Invalid Book Name");
```

Genie Ashwani

```
                        }
                        else if(ch==3)
                        {
                                System.out.print("Enter Book Name : ");
                                String name = sc.next();
                                if(map.containsKey(name))
                                {
                                        System.out.println("Price : " + map.get(name));
                                }
                                else
                                        System.out.println("Error : Invalid Book Name");
                        }
                        else if(ch==4)
                        {
                                System.out.print("Enter Book Name : ");
                                String name = sc.next();
                                if(map.containsKey(name))
                                {
                                        map.remove(name);
                                        System.out.println("Book removed");
                                }
                                else
                                        System.out.println("Error : Invalid Book Name");
                        }
                        else if(ch==5)
                        {
                                System.out.println("End");
                                System.exit(1);
                        }
                        else
                                System.out.println("Invalid choice");
                }
        }
}
```

Account Details – Menu Driven Program

Account.java:

```
public class Account
{
        private int number;
        private String name;
        private double balance;
```

```
	private String location;
	public void setNumber(int number){
		this.number = number;
	}
	public void setName(String name){
		this.name = name;
	}
	public void setBalance(double balance){
		this.balance = balance;
	}
	public void setLocation(String location){
		this.location = location;
	}
	public int getNumber(){
		return this.number;
	}
	public String getName(){
		return this.name;
	}
	public double getBalance(){
		return this.balance;
	}
	public String getLocation(){
		return this.location;
	}
}

Main.java:
import
java.util.*;class
Main
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		Map<Integer, Account> map = new HashMap<Integer, Account>();
		while(true)
		{
			System.out.println("1. Add Account");
			System.out.println("2. Update Location");
			System.out.println("3. Display Account");
			System.out.println("4. Remove Account");
			System.out.println("5. Quit");
```

Genie Ashwani

```
System.out.print("Enter your choice : ");
int ch = sc.nextInt();

if(ch==1)
{
	System.out.print("Enter Account Number : ");
	Integer number = sc.nextInt();
	if(map.containsKey(number))
	{
		System.out.println("Account already exists");
	}
	else
	{
		System.out.print("Enter Name :");
		String name = sc.next();
		System.out.print("Enter Balance :");
		double balance = sc.nextDouble();
		System.out.print("Enter Location :");
		String location = sc.next();

		Account acc = new Account();
		acc.setNumber(number);
		acc.setName(name);
		acc.setBalance(balance);
		acc.setLocation(location);

		map.put(number, acc);
		System.out.println("Account added");
	}
}
else if(ch==2)
{
	System.out.print("Enter Account Number : ");
	int number = sc.nextInt();
	if(map.containsKey(number))
	{
		System.out.print("Enter location : ");
		String location = sc.next();
		Account acc = map.get(number);
		acc.setLocation(location);
		System.out.println("Location updated");
```

```
                                }
                                else
                                        System.out.println("Error : Invalid acc-number");
                        }
                        else if(ch==3)
                        {
                                System.out.print("Enter Account Number : ");
                                int number = sc.nextInt();
                                if(map.containsKey(number))
                                {
                                        Account acc = map.get(number);
                                        System.out.println("Details : " + acc.getName() + ", " +
acc.getBalance() + ", " + acc.getLocation());
                                }
                                else
                                        System.out.println("Error : Invalid Account");
                        }
                        else if(ch==4)
                        {
                                System.out.print("Enter Account Number : ");
                                int number = sc.nextInt();
                                if(map.containsKey(number))
                                {
                                        map.remove(number);
                                        System.out.println("Account removed");
                                }
                                else
                                        System.out.println("Error : Invalid Account");
                        }
                        else if(ch==5)
                        {
                                System.out.println("End");
                                System.exit(1);
                        }
                        else
                                System.out.println("Invalid choice");
                }
        }
}
```

Comparator interface

Comparator:

- Comparator is used to order the objects of a user-defined class.
- Comparator provides multiple sorting sequences hence we can sort the elements basedon different data members, for example, rollno, name, age or anything else.

Method	Description
public int compare(Object o1, Object o2)	Compares first and second object in the list.
public boolean equals(Object obj)	Compares this object with specified object.

Compare Student objects based on Age:

Student.java:

```
class Student {
        int rollno;
        String name;
        int age;
        Student(int rollno,String name,int age){
                this.rollno=rollno;
                this.name=name;
                this.age=age;
        }
}
```

AgeComparator.java: We need to implement the Comparator interface and override the compare method to compare 2 objects and then decide to sort.

```
class AgeComparator implements Comparator {
        public int compare(Object o1,Object o2){
                Student s1=(Student)o1;
                Student s2=(Student)o2;
                if(s1.age>s2.age)
                        return 1;
                else
                        return -1;
        }
}
```

Main.java:

```
import java.util.*;
import java.io.*;
class Main
{
	public static void main(String args[]){
		ArrayList<Student> list=new ArrayList<Student>();
			list.add(new Student(101,"Vijay",23));
			list.add(new Student(106,"Ajay",27));
			list.add(new Student(105,"Jai",21));
			list.add(new Student(103, "Amar", 13));
			System.out.println("Sort by age");
			Collections.sort(list,new AgeComparator());
			System.out.println("After Sort : ");
			for(Student st : list){
				System.out.println(st.rollno+" , "+st.name+" , "+st.age);
			}
		}
	}
}
```

Compare Student objects based on Name:
NameComparator.java:

```
class NameComparator implements Comparator{
	public int compare(Object o1,Object o2){
		Student s1=(Student)o1;
		Student s2=(Student)o2;
		return s1.name.compareTo(s2.name);
	}
}
```

Sort based on Employee salary:
MySalaryComp.java:

```
class MySalaryComp implements Comparator<Empl>{
	@Override
	public int compare(Empl e1, Empl e2) {
		if(e1.getSalary() < e2.getSalary())
			return 1;
		else
			return -1;
	}
}
```

Java 8 Comparator interface

- Java 8 Comparator interface is a functional interface that contains only one abstract method.
- It is providing many static and default methods to compare different types of object elements.
- Now, we can use the Comparator interface as the assignment target for a lambda expression or method reference.

Sorting objects information using Student Age and Name:
Student.java:

```
class Student
{
	private int rollno;
	private String name;
	private int age;
	public int getRollno()
	{
		return rollno;
	}
	public void setRollno(int rollno)
	{
		this.rollno = rollno;
	}
	public String getName()
	{
		return name;
	}
	public void setName(String name)
	{
		this.name = name;
	}
	public int getAge()
	{
		return age;
	}
	public void setAge(int age)
	{
		this.age = age;
	}
}
```

Genie Ashwani

Main.java:

```
import java.util.*;
public class Code
{
	public static void main(String args[])
	{
		ArrayList<Student> al=new ArrayList<Student>();
		int nums[] = {101, 102, 103, 104};
		String names[] = {"Amar", "Swathi", "Sathya", "Harin"};
		int ages[] = {23, 30, 17, 25};

		for(int i=0 ; i<nums.length ; i++)
		{
			Student obj = new Student();
			obj.setRollno(nums[i]);
			obj.setName(names[i]);
			obj.setAge(ages[i]);
			al.add(obj);
		}

		Comparator<Student> cm1=Comparator.comparing(Student::getName);
		Collections.sort(al,cm1);
		System.out.println("Sorting by Name");
		for(Student st: al)
		{
			System.out.println(st.getRollno()+" , "+st.getName()+" , "+st.getAge());
		}

		Comparator<Student> cm2=Comparator.comparing(Student::getAge);
		Collections.sort(al,cm2);
		System.out.println("Sorting by Age");
		for(Student st: al)
		{
			System.out.println(st.getRollno()+" , "+st.getName()+" , "+st.getAge());
		}
	}
}
```

Java-8 Features

Genie Ashwani

JDK 8 Features in Java

Features are:

- Static Methods in Interface (JDK7)
- Default Methods in interface
- Functional Interface
- Lambda expression
- Method references
- forEach() method
- Stream API
- Parallel Streams
- Optional Class
- Data and Time API
- Predicates

Purpose of JDK-8:

- Using java technology, application development became easy. Billions of applications developed under java. Day by day the data increasing rapidly with the use of these applications and data processing become complex.
- JDK8 features are the solution to implement data processing techniques easily. Data processing important – for quick results

Where JDK-8 mainly works?

- After collecting the information either from data base repository or cloud, we store the information into Collection (of objects) to be processed.
- Java 8 features mainly used to process the information of Collections.

Simplifying code: JDK 8 features such as lambda expressions, method references, and default methods help reduce boilerplate code, making Java code more readable and maintainable.

Parallel processing: In order to take advantage of modern multi-core processors, JDK 8 introduced the Stream API, which enables parallel processing of collections, making it easier to write efficient and concurrent code.

Improved error handling: The Optional class was introduced to provide a more explicit way to handle null values and reduce the occurrence of NullPointerExceptions, improving code robustness.

Modernizing date and time handling: The new Date and Time API in JDK 8 was introduced to address the shortcomings of the older java.util.Date and java.util.Calendar classes, providing a more intuitive and flexible approach to date and time manipulation.

Static and Default Methods in Interface

Interface (before JDK7):

- Interface is a collection of abstract methods.
- Interface methods are by default public and abstract.
- Any class can implement interface
- Implemented class override abstract methods.

Note: The object address of implemented class assign to Interface type reference variable.
InterfaceName obj = new ImplementedClass();

```
interface Test
{
        void m1(); // public abstract
}
class Demo implements Test {
        public void m1() {
                System.out.println("m1...");
        }
}
class Main {
        public static void main(String[] args) {
                Test obj = new Demo(); // up-castingobj.m1();
        }
}
```

- Interface variables are by default public static final.
- We must initialize variables defined in interface.
- We cannot modify the variables as they are final.

```
interface Code
{
        int a = 10;  // public static final
}
class Main {
        public static void main(String[] args) {
                System.out.println("a value : " + Code.a);
                Code.a = 20 ; // Error : final variable cannot be modified
        }
}
```

Genie Ashwani

Static and Default Methods

Interface (since jdk7):

- Interface allowed to define static methods from jdk7
- Static methods can access using identity of interface.

Note: Static methods are used to define the common functionality of objects which are implemented from that interface.

```
interface CreditCard {
	String cartType = "VISA-Platinum";
	static void benefits(){
		System.out.println("Benefits on Flying, Dining and more");
	}
}
```

Default Methods:

- Defining a method with default keyword.
- We can access Default methods through object reference.

Note: Default methods allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

```
interface Vehicle{
	default String airBags(){
		return "Two airbags";
	}
	default String alarmOn(){
		return "at speed of 100";
	}
	int maxSpeed();
}
class Alto implements Vehicle{
	public int maxSpeed(){
		return 160;
	}
}
class Swift implements Vehicle{
	public int maxSpeed(){
		return 220;
	}
}
```

Genie Ashwani

Functional Interface in Java

Functional Interface:

- Interface that accepts only one abstract method.
- We must define with annotation @FunctionalInterface.
- Functional Interface allow static and default methods.

```
@FunctionalInterface
interface Test
{
        void m1();
        void m2();  // Error :
}
```

Static Methods and Default Methods in Functional Interface:

```
@FunctionalInterface
interface First
{
        static void m1(){
                System.out.println("Static method");
        }
        default void m2(){
                System.out.println("Default method");
        }
        void m3();
}
class Second implements First
{
        public void m3(){
                System.out.println("Instance method");
        }
}
class Main
{
        public static void main(String[] args)
        {
                First obj = new Second();
                First.m1();
                obj.m2();
                obj.m3();
        }
}
```

Genie Ashwani

Lambda Expressions

Lambda Expression: Lambda expression is a simplest form of Functional interface implementation.

In how many ways we can implement a Functional Interface:

1. Through class
2. Through anonymous inner class
3. Through lambda expression

1. **Implement interface using class:**

```
@FunctionalInterface
interface First {
        void fun();
}
class Second implements First {
        public void fun(){
                System.out.println("fun...");
        }
}
class Main {
        public static void main(String[] args) {
                First obj = new Second();
                obj.fun();
        }
}
```

2. **Through Anonymous inner class:** Defining a class without identity is called Anonymous inner class. We always define anonymous class inside a method.

```
interface Test {
        void fun();
}
class Main {
        public static void main(String[] args) {
                Test obj = new Test() {
                        public void fun() {
                                System.out.println("Anonymous fun");
                        }
                };
                obj.fun();
        }
}
```

Genie Ashwani

Through Lambda expression:

- Expression is a line of code.
- Lambda expression is the implementation of Functional Interface in a short format

```
@FunctionalInterface
interface Test
{
        void fun();
}
class Main
{
        public static void main(String[] args) {
                Test obj = () -> System.out.println("Lambda fun");
                obj.fun();
        }
}
```

If the method not taking any parameter:

```
() -> expression
```

If the method taking only one parameter:

```
parameter -> expression
```

If the method taking more than one parameter:

```
(parameter1, parameter2) -> expression
```

Lambda expression as block:

- Expressions immediately return a value, and they cannot contain variables, assignments or statements such as if or for.
- In order to do more complex operations, a code block can be used with curly braces.
- If the lambda expression needs to return a value, then the code block should have a return statement.

```
(parameter1, parameter2) -> {
        stat-1;
        stat-2;
        stat-3;
        return
}
```

Lambda expression with arguments:

- Lambda expression can take arguments based on the signature of method defined in functional interface.
- No need to specify the data types while representing the arguments in lambda expression.

```
@FunctionalInterface
interface Calc
{
	void add(int x, int y);
}
class Main
{
	public static void main(String[] args)
	{
		Calc obj = (x, y) -> System.out.println("Sum : " + (x+y));
		obj.add(5,3);
		obj.add(10,20);
	}
}
```

Lambda expression with return values: Lambda expression automatically returns the value which is evaluated in expression. We need to specify the return type in Functional Interface specification.

```
@FunctionalInterface
interface Calc
{
	int add(int x, int y);
}
class Main
{
	public static void main(String[] args)
	{
		Calc obj = (x, y) -> x+y;

		System.out.println("Sum : " + obj.add(5,3));
		System.out.println("Sum : " + obj.add(10,20));
	}
}
```

Genie Ashwani

Method References in Java

Method references:

- It is JDK8 feature.
- It is used to refer any method of functional interface easily.
- Any method definition can be assigned to functional interface method identity and access the using its identity.

Method reference to static method as follows:

```
@FunctionalInterface
interface Test {
	void abc();
}
class Demo {
	static void fun() {
		System.out.println("fun...");
	}
}
class Main {
	public static void main(String[] args) {
		Test obj = Demo::fun;
		obj.abc();
	}
}
```

Method reference to an instance method:

```
@FunctionalInterface
interface Test {
	void abc();
}
class Demo {
	void fun() {
		System.out.println("fun...");
	}
}
class Main {
	public static void main(String[] args) {
		Test obj = new Demo()::fun;
		obj.abc();
	}
}
```

Genie Ashwani

forEach() method

forEach() method:

- forEach() introduced in JDK8 to iterate the collection easily.
- forEach() defined in both Iterable interface and in Stream interface.
- forEach() method is a Default method.

```
default void forEach(Consumer<super T>action)
```

Note: forEach() method takes a single parameter which is a functional interface. So, you can pass lambda expression or method reference as input.

Iterating List

Defining a List:

```
List<String> names = Arrays.asList("C", "Java", "Python");
```

using lambda expression:

```
names.forEach(name->System.out.println(name));
```

using method reference:

```
names.forEach(System.out::println);
```

Iterating Set

Defining a Set:

```
Set<String> uniqueNames = new HashSet<>(Arrays.asList("C", "C++", "Java"));
```

using lambda expression:

```
uniqueNames.forEach(name->System.out.println(name));
```

using method reference:

```
uniqueNames.forEach(System.out::println);
```

Iterating Map

Defining a Map:

```
Map<Integer, String> namesMap = new HashMap<>();
namesMap.put(1, "Java");
namesMap.put(2, "JDBC");
namesMap.put(3, "JSP");
```

Iterate map:

```
namesMap.forEach((key, value) -> System.out.println(key + " " + value));
```

Iterating Map using entrySet:

```
namesMap.entrySet().forEach(entry -> System.out.println(
  entry.getKey() + " " + entry.getValue()));
```

Genie Ashwani

Iterate List:

```
import
java.util.*; import
java.util.*;    class
Main
{
        public static void main(String[] args)
        {
                List<Integer> list = new
                ArrayList<>(Arrays.asList(10,20,30,40,50));list.forEach(x-
                >System.out.println(x));
        }
```

Iterate Set:

```
import
java.util.*; import
java.util.*;    class
Main
{
        public static void main(String[] args)
        {
                List<Integer> list = new ArrayList<>(Arrays.asList(10,20,30,40,50));
                Set<Integer> set = new HashSet<>(list);
                set.forEach(x->System.out.println(x));
        }
}
```

Iterate Map:

```
import
java.util.*; import
java.util.*;    class
Main
{
        public static void main(String[] args)
        {
                Map<Integer,String> map = new HashMap<>();
                map.put(1, "Java");
                map.put(2, "Servlets");
                map.put(3, "JSP");
                map.forEach((k, v)->System.out.println(k + " = " + v));
        }
}
```

Genie Ashwani

Practice Programs on forEach()

Printing all elements of an ArrayList:

```
ArrayList<String> list = new ArrayList<>(Arrays.asList("apple", "banana", "cherry"));
list.forEach(System.out::println);
```

Doubling the values in a LinkedList:

```
LinkedList<Integer> numbers = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
numbers.forEach(n -> System.out.println(n * 2));
```

Checking if any string in a HashSet starts with "A":

```
HashSet<String> set = new HashSet<>(Arrays.asList("Apple", "Banana", "Cherry"));
set.forEach(s -> {
        if (s.startsWith("A")) {
                System.out.println(s);
        }
});
```

Removing all even numbers from an ArrayList:

```
ArrayList<Integer> numbers = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));
numbers.forEach(n -> {
        if (n % 2 == 0) {
                numbers.remove(n);
        }
});
```

Converting all strings in a TreeSet to uppercase:

```
TreeSet<String> set = new TreeSet<>(Arrays.asList("apple", "banana", "cherry"));
set.forEach(s -> {
        String upperCase = s.toUpperCase();
        System.out.println(upperCase);
});
```

Checking if a LinkedList contains a specific value

```
LinkedList<Integer> numbers = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));int
target = 3;
 numbers.forEach(n -> {
         if (n == target) {
                System.out.println("Found: " + n);
        }
});
```

Genie Ashwani

Removing all elements from a HashSet that are less than 10

```
HashSet<Integer> numbers = new HashSet<>(Arrays.asList(5, 10, 15, 20));
numbers.forEach(n -> {
        if (n < 10) {
                numbers.remove(n);
        }
});
```

Concatenating all strings in an ArrayList:

```
ArrayList<String> strings = new ArrayList<>(Arrays.asList("Hello", " ", "World", "!"));
StringBuilder sb = new StringBuilder();
strings.forEach(sb::append);
System.out.println(sb.toString())
.
```

Printing the square of each element in a LinkedList:

```
LinkedList<Integer> numbers = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
numbers.forEach(n -> System.out.println(n * n));
```

Counting the number of occurrences of a specific string in an ArrayList:

```
ArrayList<String> strings = new ArrayList<>(Arrays.asList("apple", "banana", "cherry",
"apple"));
String target = "apple";
int count = 0;
strings.forEach(s -> {
        if (s.equals(target)) {
                count++;
        }
});
System.out.println("Occurrences of \"" + target + "\": " + count);
```

Genie Ashwani

Stream API

Stream API:

- Stream is a flow of data.
- Stream API providing pre-defined functionality by which we can filter the data easily.
- We always create Streams to Collection objects and filter the data stored in collections.

Note: Stream is a not a data structure hence it will not hold any information. Stream will not change the collection object. It just processes the elements of object by without modifying it.

Creating Stream to List: stream() method returns the stream of any collection object

```
Stream<Integer> st = list.stream();
```

forEach() method in Stream interface:

- Stream API related interfaces and classes belongs to java.util.stream package.
- forEach() method belongs to Stream class also.
- We invoke the forEach() method on Stream object to display the data.

Display Stream information using forEach() and Lambda:

```
import  java.util.*;
import java.util.stream.*;
class Main {
	public static void main(String[] args) {
		List<Integer> list = new ArrayList<>(Arrays.asList(10,20,30,40,50));
		Stream<Integer> st = list.stream();
		st.forEach(x->System.out.println(x));
	}
}
```

Stream and display the list in single line:

```
import  java.util.*;
import java.util.stream.*;
class Main {
	public static void main(String[] args) {
		List<Integer> list = new
		ArrayList<>(Arrays.asList(10,20,30,40,50));list.stream().forEach(x-
		>System.out.println(x));
	}
}
```

map() and filter() methods of Stream

map():

- It takes a lambda expression as its only argument, and uses it to change every individual element in the stream.
- Its return value is a new Stream object containing the changed elements.

Creating a Stream:

```
List<Integer> myList = new ArrayList<Integer>();
myList.add(1);
myList.add(5)
;
myList.add(8)
;
```

We can create streams for arrays as follows:

```
Integer[] myArray = {1, 5, 8};
Stream<Integer> myStream = Arrays.stream(myArray);
```

Map to convert all elements in an array of strings to uppercase:

```
String[] myArray = new String[]{"harin", "satya", "annie", "amar"};
Stream<String> myStream = Arrays.stream(myArray);
Stream<String> myNewStream = myStream.map(s -> s.toUpperCase());
```

Practice Programs

Convert a list of strings to uppercase:

```
List<String> strings = Arrays.asList("hello", "world", "java");
List<String> upperCaseStrings = strings.stream()
                                .map(String::toUpperCase)
                                .collect(Collectors.toList())
;System.out.println(upperCaseStrings);
```

Square all the numbers in a list of integers:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
List<Integer> squaredNumbers = numbers.stream()
                               .map(num -> num * num)
                               .collect(Collectors.toList())
;System.out.println(squaredNumbers);
```

Extract the lengths of strings in a list:

```
List<String> fruits = Arrays.asList("apple", "banana", "orange");
List<Integer> lengths = fruits.stream()
                        .map(String::length)
                        .collect(Collectors.toList())
;System.out.println(lengths);
```

Get the first character of each string in a list:

```
List<String> words = Arrays.asList("cat", "dog", "bird");
List<Character> firstCharacters = words.stream()
                                  .map(s -> s.charAt(0))
                                  .collect(Collectors.toList());
System.out.println(firstCharacters);
```

filter():

- filter() method takes lambda expression as input and return boolean value.
- If filter() method returns true, then the element enter into resultant stream.
- filter() method returns stream after filtering the data.

Program to filter the Strings starts with "s" in a list:

```
import  java.util.*;
import java.util.stream.*;
class Main
{
        public static void main(String[] args) { List<String>
                list = new ArrayList<String>();
                list.add("Java");
                list.add("JDBC");
                list.add("Servlets");
                list.add("JSP");
                list.add("Spring");
                list.add("Hibernate");

                Stream<String> st = list.stream();
                Stream<String> res = st.filter(s->s.startsWith("S"));
                res.forEach(System.out::println);
        }
}
```

Write the above filtering logic in single line:

```
list.stream().filter(s-
```

Display string whose length is 4:

```
import  java.util.*;
import java.util.stream.*;
class Main {
	public static void main(String[] args)
	{
		String[] arr = {"Java", "JDBC", "Servlets", "JSP", "Spring"};
		List<String> list = new ArrayList<String>();
		for(int i=0 ; i<arr.length ; i++)
			list.add(arr[i]);

		list.stream().filter(s->s.length()==4).forEach(System.out::println);
	}
}
```

Program to display only even numbers in the list using stream api:

```
import  java.util.*;
import java.util.stream.*;
class Main {
	public static void main(String[] args) {
		int[] arr = {5, 2, 8, 9, 3, 7, 1, 4, 6};
		List<Integer> list = new ArrayList<Integer>();
		for(int x : arr)
			list.add(x);
		System.out.println("List is : " + list);
		System.out.println("Even numbers list is : ");
		list.stream().filter(s-
		>s%2==0).forEach(System.out::println);
	}
}
```

Collectors Class

Collectors: Collectors is a final class. It provides methods to collect filtered elements into various collections, and performing operations on collected data such as counting elements, reverse, sort etc.

Store into List:

```
List<Integer> integers = Arrays.asList(1,2,3,4,5,6,6);
integers.stream().map(x ->
x*x).collect(Collectors.toList());

output: [1,4,9,16,25,36,36]
```

Store into Set:

```
List<Integer> integers = Arrays.asList(1,2,3,4,5,6,6);
integers.stream().map(x -> x*x).collect(Collectors.toSet());

output: [1,4,9,16,25,36]
```

Store into Specific Collection:

```
List<Integer> integers = Arrays.asList(1,2,3,4,5,6,6);
integers
    .stream()
    .filter(x -> x >2)
    .collect(Collectors.toCollection(LinkedList::new));

output: [3,4,5,6,6]
```

Counting elements:

```
List<Integer> integers = Arrays.asList(1,2,3,4,5,6,6);
Long collect = integers
                .stream()
                .filter(x -> x <4)
                .collect(Collectors.counting());

Output: 3
```

Finding minimum value: minBy()

```
List<Integer> integers = Arrays.asList(1,2,3,4,5,6,6);
integers
    .stream()
    .collect(Collectors.minBy(Comparator.naturalOrder()))
    .get();

Output: 1
```

Genie Ashwani

Practice programs using map() and filter()

Filter and map positive integers to their squares:

```
List<Integer> numbers = Arrays.asList(1, -2, 3, -4, 5);
List<Integer> squaresOfPositiveNumbers = numbers.stream()
                                        .filter(n -> n > 0)
                                        .map(n -> n * n)
                                        .collect(Collectors.toList())
;System.out.println(squaresOfPositiveNumbers);
```

Filter and map strings to their uppercase versions:

```
List<String> words = Arrays.asList("apple", "Banana", "orange", "Grapes");
List<String> upperCaseWords = words.stream()
                              .filter(s -> s.length() > 3)
                              .map(String::toUpperCase)
                              .collect(Collectors.toList())
;System.out.println(upperCaseWords);
```

Filter and map a list of objects:

```
List<Person> persons = Arrays.asList(
    new Person("John", 25),
    new Person("Alice", 30),
    new Person("Bob", 40)
);
List<String> namesAboveThirty = persons.stream()
                                .filter(p -> p.getAge() > 30)
                                .map(Person::getName)
                                .collect(Collectors.toList())
;System.out.println(namesAboveThirty);
```

Filter and map a list of strings to their lengths:

```
List<String> words = Arrays.asList("apple", "banana", "orange", "grapes");
List<Integer> wordLengths = words.stream()
                            .filter(s -> s.startsWith("a"))
                            .map(String::length)
                            .collect(Collectors.toList())
;System.out.println(wordLengths);
```

Genie Ashwani

Filter and map a list of strings to their lengths, removing duplicates:

```
List<String> words = Arrays.asList("apple", "banana", "orange", "grapes", "apple");
List<Integer> distinctWordLengths = words.stream()
                                    .filter(s -> s.length() > 4)
                                    .map(String::length)
                                    .distinct()
                                    .collect(Collectors.toList())
;System.out.println(distinctWordLengths);
```

Filter and map a list of integers, then calculate their average:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
double average = numbers.stream()
                    .filter(n -> n % 2 == 0)
                    .mapToInt(Integer::intValue)
                    .average()
                    .orElse(0);
System.out.println(average);
```

Filter and map a list of strings to their lengths, then find the maximum length:

```
List<String> words = Arrays.asList("apple", "banana", "orange", "grapes");
int maxLength = words.stream()
                .filter(s -> s.length() > 5)
                .mapToInt(String::length)
                .max()
                .orElse(0);
System.out.println(maxLength);
```

Filter and map a list of strings to their uppercase versions, then sort them:

```
List<String> words = Arrays.asList("apple", "banana", "orange", "grapes");
List<String> sortedUpperCaseWords = words.stream()
                                    .filter(s -> !s.isEmpty())
                                    .map(String::toUpperCase)
                                    .sorted()
                                    .collect(Collectors.toList())
;System.out.println(sortedUpperCaseWords);
```

Genie Ashwani

Parallel Streaming

Parallel Streaming: Parallel streaming is a feature introduced in Java 8 as part of the Stream API. It allows you to perform stream operations concurrently on multi-core processors, effectively dividing the data into smaller chunks and processing them in parallel threads.

```
import
java.util.LinkedList;import
java.util.List; public class
Demo
{
	public static void main(String[] args)
	{
		List<String> colors = new LinkedList<>();
		colors.add("Red");
		colors.add("Green");
		colors.add("Blue");
		colors.add("Yellow")
		;
		colors.add("Purple");

		colors.parallelStream()
		.forEach(System.out::println);
	}
}
```

Parallel stream with HashMap:

```
import java.util.HashMap;
import java.util.Map;
public class Demo
{
	public static void main(String[] args)
	{
		Map<Integer, String> map = new HashMap<>();
		map.put(1, "One");
		map.put(2, "Two");
		map.put(3, "Three");
		map.put(4, "Four");
		map.put(5, "Five");

		map.entrySet().parallelStream()
		.forEach(entry -> System.out.println(entry.getKey() + ": " + entry.getValue()));
	}
}
```

Genie Ashwani

Parallel stream with Array:

```
import java.util.Arrays;
public class ParallelStreamArrayExample
{
	public static void main(String[] args)
	{
		String[] names = {"Alice", "Bob", "Charlie", "David", "Eve"};
		Arrays.stream(names).parallel().forEach(System.out::println);
	}
}
```

forEachOrdered():

```
import java.util.*;
public class Demo
{
	public static void main(String[] args)
	{
		List<Integer> list = Arrays.asList(2, 4, 6, 8, 10);
		list.stream().parallel().forEach( System.out::println );
		list.stream().parallel().forEachOrdered( System.out::println );
	}
}
```

Parallel Stream for Array Sum:

```
import java.util.Arrays;
public class ParallelStreamArraySum
{
	public static void main(String[] args)
	{
		int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
		int sum =
		Arrays.stream(numbers).parallel().sum();
		System.out.println("Sum of numbers: " + sum);
	}
}
```

Optional Class

Optional class:

- In Java 8, the concept of Optional was introduced to handle the case where a method might return a non-null value or a null value.
- Optional is a container object that may or may not contain a non-null value.
- It helps to avoid null pointer exceptions and allows developers to write more concise and expressive code.

Display the value of string only if it is not null:

```
import java.util.Optional;
public class Code
{
	public static void main(String[] args)
	{
		String value = "Hello";
		Optional<String> optionalValue = Optional.of(value);
		System.out.println(optionalValue.get()); // Output: Hello
	}
}
```

Display the default value in place of String if the string value is NULL:

```
import java.util.Optional;
public class Code
{
	public static void main(String[] args)
	{
		String value = null;
		Optional<String> optionalValue = Optional.ofNullable(value);
		System.out.println(optionalValue.orElse("Default")); // Output: Default
	}
}
```

Check the String has null value or Not:

```
import java.util.Optional;
public class Code {
	public static void main(String[] args)
	{
		String value = null;
		Optional<String> optionalValue = Optional.ofNullable(value);
		System.out.println(optionalValue.isPresent()); // Output: false
	}
}
```

Creating an Optional with a Non-null Value:

```
import java.util.Optional;
public class Code
{
	public static void main(String[] args)
	{
		String value = "Hello, World!";
		Optional<String> optionalValue = Optional.of(value);

		// If the value is present, print it
		optionalValue.ifPresent(System.out::println);
	}
}
```

Creating an Optional with a Nullable Value:

```
import java.util.Optional;
public class Code
{
	public static void main(String[] args)
	{
		String value = null;
		Optional<String> optionalValue = Optional.ofNullable(value);
		optionalValue.ifPresentOrElse(System.out::println, () ->
System.out.println("Default Value"));
	}
}
```

Performing an Action on the Value if Present:

```
import java.util.Optional;
public class Code
{
	public static void main(String[] args)
	{
		Integer number = 42;
		Optional<Integer> optionalNumber = Optional.of(number);
		optionalNumber.ifPresent(num -> {
		    int incrementedValue = num + 10;
		    System.out.println("Incremented value: " + incrementedValue);
		});
	}
}
```

Genie Ashwani

Predicates

Predicates in Java 8 are an essential part of the java.util.function package, designed to handle functional-style operations. Here are 10 simple points to understand Predicates in Java 8:

Functional Interface: It is a functional interface that takes an argument of a specific type and returns a boolean value (true or false).

Single Abstract Method (SAM): As a functional interface, Predicate has a single abstract method named test(T t), where T is the type of the input argument.

Test Condition: The test(T t) method is responsible for evaluating the condition specified by the Predicate. If the condition is met for the input t, the method returns true; otherwise, it returns false.

Filtering Collections: Predicates are commonly used to filter elements from collections. By passing a Predicate to the filter() method of Java 8 streams or collections, you can selectively retain elements that satisfy the condition.

Chaining Predicates: Predicates can be combined using logical operators like and(), or(), and negate() to create more complex conditions for filtering.

Default Methods: The Predicate interface provides default methods like and(), or(), and negate()to enable easy composition of predicates without requiring manual implementation.

Avoiding Null Checks: Predicates can be used to check for null values or other conditions that could cause null pointer exceptions, providing a safer and more readable alternative to traditional null checks.

Java Collections API: Java 8 introduced several methods in the Collections API that accept Predicate as an argument, such as removeIf(), making it easier to perform conditional element removal from collections.

Lambda Expressions: Predicates can be defined using lambda expressions, allowing for concise and expressive code when dealing with conditional checks.

Filtering even numbers from a list of integers:

```
import
java.util.Arrays;import
java.util.List;
import java.util.function.Predicate;
public class Code
{
```

```
        public static void main(String[] args)
        {
                List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                Predicate<Integer> isEven = num -> num % 2 == 0;
                numbers.stream().filter(isEven).forEach(System.out::println);
        }
}
```

Filtering strings with a specific length from a list of strings:

```
import
java.util.Arrays;import
java.util.List;
import java.util.function.Predicate;
public class Code
{
        public static void main(String[] args)
        {
                List<String> words = Arrays.asList("apple", "banana", "orange", "grape", "kiwi");
                Predicate<String> hasSpecificLength = word -> word.length() == 5;
                words.stream().filter(hasSpecificLength).forEach(System.out::println);
        }
}
```

Filtering names starting with a specific letter from a list of names:

```
import
java.util.Arrays;import
java.util.List;
import java.util.function.Predicate;
public class Code
{
        public static void main(String[] args)
        {
                List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David", "Eva");
                Predicate<String> startsWithA = name -> name.startsWith("A");
                names.stream().filter(startsWithA).forEach(System.out::println);
        }
}
```

Filtering positive numbers from an array of doubles:

```
import java.util.Arrays;
import java.util.function.Predicate;
public class Code
{
        public static void main(String[] args)
```

```
        {
                double[] numbers = { 1.2, -3.4, 5.6, -7.8, 9.0 };
                Predicate<Double> isPositive = num -> num > 0;
                Arrays.stream(numbers).filter(isPositive).forEach(System.out::println);
        }
}
```

Checking if any element in the list satisfies the predicate:

```
import
java.util.Arrays;import
java.util.List;
import java.util.function.Predicate;
public class Code
{
        public static void main(String[] args)
        {
                List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                Predicate<Integer> isEven = num -> num % 2 == 0;
                boolean anyEven = numbers.stream().anyMatch(isEven);
                System.out.println("Any even number? " + anyEven);
        }
}
```

Chaining predicates to filter numbers greater than 5 and even:

```
import
java.util.Arrays;import
java.util.List;
import java.util.function.Predicate;
public class Code
{
        public static void main(String[] args)
        {
                List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
                Predicate<Integer> isEven = num -> num % 2 == 0; Predicate<Integer>
                isGreaterThan5 = num -> num > 5;

                numbers.stream().filter(isEven.and(isGreaterThan5)).forEach(System.out::println)
                ;
        }
```

Date and Time API

- The Date and Time API (JSR 310) providing a comprehensive set of classes to handle date, time, and duration operations.
- JSR 310 stands for "Java Specification Request 310," and it is the formal request to addthe Date and Time API to the Java Standard Edition (Java SE).
- It is the solution to problems in java.util.Date and java.util.Calendar classes.
- The classes in the Date and Time API are immutable, meaning their values cannot be changed once created, ensuring thread safety in multi-threaded environments.
- The API is part of the java.time package and includes classes such as LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Duration, Period, and more.

Getting the current date:

```
import java.time.LocalDate;
public class Code
{
	public static void main(String[] args)
	{
		LocalDate currentDate = LocalDate.now();
		System.out.println("Current Date: " + currentDate);
	}
}
```

Getting the current time:

```
import java.time.LocalTime;
public class Code
{
	public static void main(String[] args)
	{
		LocalTime currentTime = LocalTime.now();
		System.out.println("Current Time: " + currentTime);
	}
}
```

Parsing a date from a string:

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
public class Code
{
	public static void main(String[] args)
	{
		String dateString = "2023-07-24";
		LocalDate date = LocalDate.parse(dateString);
```

Genie Ashwani

```
                System.out.println("Parsed Date: " + date);
        }
}
```

Formatting time with seconds:

```
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
public class Code
{
        public static void main(String[] args)
        {
                LocalTime time = LocalTime.of(12, 34, 56);
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
                String formattedTime = time.format(formatter);
                System.out.println("Formatted Time: " + formattedTime);
        }
}
```

Adding days to a date:

```
import java.time.LocalDate;
public class Code
{
        public static void main(String[] args)
        {
                LocalDate date = LocalDate.now();
                LocalDate futureDate = date.plusDays(10);
                System.out.println("Future Date: " + futureDate);
        }
}
```

Checking if a date is before or after another date:

```
import java.time.LocalDate;
public class Code
{
        public static void main(String[] args)
        {
                LocalDate date1 = LocalDate.of(2023, 7, 24);
                LocalDate date2 = LocalDate.of(2023, 12, 31);
                System.out.println("Is date1 before date2? " + date1.isBefore(date2));
                System.out.println("Is date1 after date2? " + date1.isAfter(date2));
        }
}
```

Genie Ashwani

Formatting a date to a custom string:

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
public class Code
{
	public static void main(String[] args)
	{
		LocalDate date = LocalDate.of(2023, 7, 24);
		DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
		String formattedDate = date.format(formatter);
		System.out.println("Formatted Date: " + formattedDate);
	}
}
```

Calculating the difference between two dates:

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
public class Code
{
	public static void main(String[] args)
	{
		LocalDate date1 = LocalDate.of(2023, 7, 24);
		LocalDate date2 = LocalDate.of(2023, 12, 31);

		long daysDifference = ChronoUnit.DAYS.between(date1, date2);
		System.out.println("Days Difference: " + daysDifference);
	}
}
```

Checking if a year is a leap year:

```
import java.time.Year;
public class Code
{
	public static void main(String[] args)
	{
		int year = 2024;
		boolean isLeapYear = Year.of(year).isLeap();
		System.out.println(year + " is a leap year? " + isLeapYear);
	}
}
```

Summary

Functional Interfaces:

- Functional interfaces have only one abstract method and are used to enable lambda expressions and method references.
- JDK 8 introduced the @FunctionalInterface annotation to indicate that an interface is intended to be used as a functional interface.
- Common functional interfaces like Predicate, Function, Consumer, and Supplier are available in the java.util.function package.

Lambda Expressions:

- They provide a concise way to define single-method interfaces (functional interfaces).

Default Methods:

- Default methods are methods defined in interfaces with a default implementation.
- They were introduced to support backward compatibility when adding new methods to existing interfaces.
- Default methods allow developers to add new functionality to interfaces without breaking the implementations of existing classes.

forEach()

- forEach() is a terminal operation in Java 8 streams used to perform an action on each element of the stream.
- It accepts a lambda expression or method reference that defines the action to be executed for each element.
- forEach() does not return any value; it is mainly used for performing side effects.

forEachOrdered()

- forEachOrdered() is similar to forEach(), but it guarantees that the elements are processed in the order they appear in the stream.
- It is mainly useful for parallel streams when you want to ensure ordered processing regardless of parallelization.

Stream()

- Stream is an interface introduced in Java 8 that represents a sequence of elements that can be processed in a functional-style manner.

Stream API:

- The Stream API is a powerful addition to JDK 8, allowing functional-style operations on collections and arrays.
- The Stream API includes operations like map(), filter(), reduce(), forEach(), and collect().
- Parallel streams leverage multiple threads to enhance performance when processing large datasets.

Genie Ashwani

map():

- map() is an intermediate operation that transforms each element of a stream into another object using the provided function.
- It takes a Function as an argument, which defines the mapping from one object to another.

filter():

- filter() is an intermediate operation used to eliminate elements from a stream based on a specified condition.
- It takes a Predicate as an argument, which defines the condition for filtering the elements.
- The result of the filter() operation is a new stream containing only the elements that satisfy the given condition.

reduce():

- reduce() is a terminal operation that aggregates the elements of a stream into a single result.
- It takes two arguments: an identity value and a BinaryOperator, which combines two elements into a single result.

Predicate():

- Predicate is a functional interface from the java.util.function package that represents a boolean-valued function of one argument.
- It is often used in streams to define conditions for filtering elements.

Optional:

- Optional is a container object that may or may not contain a non-null value.
- It was introduced to help handle the problem of null references and avoid null pointer exceptions.
- Optional provides methods like isPresent(), orElse(), and ifPresent() to work with possibly absent values in a more explicit and safer way.

Date and Time API (JSR 310):

- JDK 8 introduced a new Date and Time API based on JSR 310, addressing various issuesin the legacy java.util.Date and java.util.Calendar classes.
- The new API is immutable, thread-safe, and provides better representation and manipulation of dates, times, and durations.
- Classes like LocalDate, LocalTime, LocalDateTime, and ZonedDateTime provide easy-to-use representations of dates and times without time zone hassles.

Collections – Interview Questions

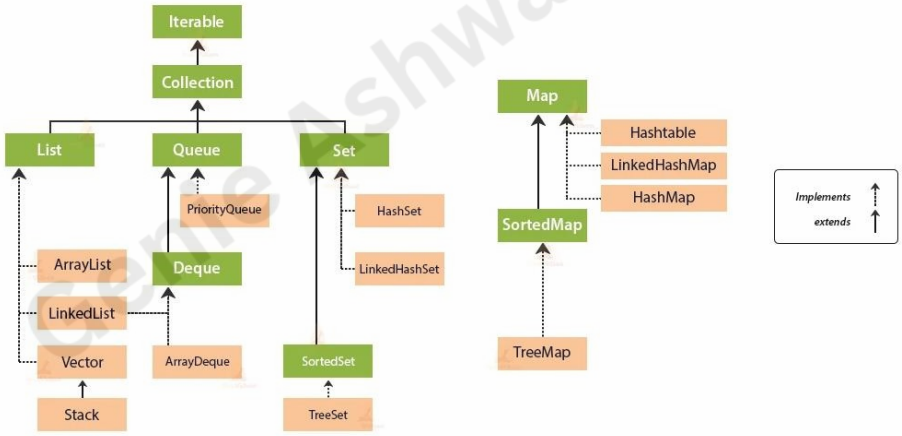
1. **Define Variable and Method?**
   - **Variable:** Is used to Store the data
     double balance ;
   - **Method:** Is used to Process the data
     getBalance();
     setBalance();

2. **Define Array?**
   - **Array:** Is a set of similar data elements
     long[] accNums;

3. **Define Object?**
   - Object is a set of dis-similar data elements. For example, Student details, Employee details, Customer details etc.

4. **Define Collection?**
   - Collection is set of Objects. Set of Student details, Employee details.

5. **Collection Interface and Classes:**



6. **Array v/s ArrayList**
   - Array is Static (Fixed size)
   - ArrayList is Dynamic (No Fixed Size)

7. **Define List:**
   - List is ordered
   - List allow duplicates
   - List is index based
8. **Define Set:**

- Set is not ordered
- Set doesn't allow duplicates
- Set is not index based

9. Define Map:

- Map store elements using keys
- Keys must be unique
- Values can be duplicated.

10. ArrayList v/s Vector:

- Vector is synchronized by default
- ArrayList is not synchronized by default

11. Vector v/s Stack:

- Vector is index based
- Stack follows LIFO (Last In First Out)

12. ArrayList v/s LinkedList:

- **In ArrayList:** Accessing elements is faster (index based)
  Occupies Less memory.
  Insertions and Deletions are Slower
- **In LinkedList:** Insertions and Deletions are faster (no shifting of elements)
  Occupies More memory.

13. Define HashSet, LinkedHashSet and TreeSet?

- **HashSet:** doesn't maintains insertion order of elements
- **LinkedHashSet:** maintains insertion order of elements
- **TreeSet:** maintains sorted order of elements

14. Define HashMap, LinkedHashMap and TreeMap?

- **HashMap:** doesn't maintain insertion order of elements
- **LinkedHashMap:** maintains insertion order of elements
- **TreeMap:** maintains sorted order of elements using keys.

15. Define Hashtable?

- Hashtable store elements using keys.
- Hashtable doesn't allow null keys.

16. Explain null values in collections?

- List is allowed to store any number of null values
- Set is allowed to store only one null value
- In map – we can store one null key but any number of null values

17. Which of the Collections synchronized by default?

- Only legacy collection classes synchronized by default
  Examples: Vector, Stack and Hashtable

18. Define PriorityQueue?

- Store elements in insertion order but display elements using their priority

19. Differentiate Iterator and ListIterator?

- **Iterator:** Process elements one by one in forward direction
  Iterator is not index based
  Iterator can process List, Set and Map
- **ListIterator:** Process elements using specified index
  Iterator can process only List(Set and Map not index based)

20. Define Boxing and Unboxing?

- **Boxing:** Conversion of primitive data to Object
- **Unboxing:** Conversion of Object type data to primitive

21. Define Auto-Boxing and Auto Unboxing?

- **Auto-Boxing:** Auto conversion from primitive to Object
- **Auto Unboxing:** Auto conversion from object to primitive
  Note: Auto Boxing & Auto Unboxing since jdk5

22. Define Comparable and Comparator?

- Comparable and Comparator is used sort record type objects.
- Comparable sort objects based on single value like id or name or location
- Comparator sort objects based on multiple values like id-name, name-location,id-location.

23. Define Collections class?

- Collections is a class belongs to util package
- Collections class providing searching, sorting and conversion methods to process collection objects easily.

24. Collection v/s Collection with Generics:

- **Collection:** Collection accepts any type of object.
  No type safety.
- **Collection with Generics:** Collection with Generics allow to store specified typeof Objects. Generics for type safety.

25. How to convert ArrayList to Array?

- Arrays.asList(item);

JDK8 Features – Interview Questions

1. **What are JDK8 features?**
   - Static and Default methods in interface
   - Functional Interface, Lambda expressions & Method references
   - forEach() and forEachOrdered() method
   - Stream API, Collectors class
   - Predicates
   - Time and Data API
   - Optional Class

2. **What is the use of JDK8 features?**
   - JDK8 features are used to process the information quickly and with short code.
   - Streaming the data is used to perform operations like searching, sorting, filtering, parallel processing etc.

3. **What is an interface in JDK8?**
   - Since JDK8, interface allow static and default methods along with abstract methods final variables.

4. **Explain static methods in interface?**
   - Defining a method with static keyword. Static represents common functionality of interface. Static methods can access using identity of interface.

5. **How to define default methods?**
   - Define a method using default keyword. We can access default methods using object reference.

6. **What is Functional Interface?**
   - An interface with only one abstract method.
   - It is recommended to define Functional Interface using @FunctionalInterface annotation.
   - It allows any number of static and default methods.

7. **What is anonymous inner class?**
   - Define a class without identity inside the method. We often implement interfacesas anonymous inner classes.

8. **What is lambda expression?**
   - Easy implementation of functional interface.
   - Lambda expression is an object and address are assigned to FunctionalInterface reference variable.

Genie Ashwani

9. Define Method references?

- Method references are used to refer static or instance method of a class.
- It is mainly used to refer a method of Functional interface.

10. How to create reference to static method?

- ClassName::MethodName

11. How to create reference to instance method?

- new ClassName()::MethodName

12. Define forEach() method?

- It is used to iterate element of collection or stream
- It is belonging to Iterable interface & Stream interface

13. How can we display the elements of List in jdk8?

- Lambda expression:
  - list.forEach(n->System.out.println(n));
- Method reference:
  - list.forEach(System.out::println);

14. What is Stream API?

- Stream is a flow of data.
- We can create streams to collection object data to process elements.
- Stream enables us to combine multiple operations on data to get desired result.
- Jjva.util.stream package is providing Stream interface

15. How to create Stream for List?

- Stream<E> s = list.stream();

16. How to sort list elements using stream api?

- list.stream().sorted().forEach(System.out::println);

17. Define Collectors class?

- Collectors class belongs to java.util. package.
- Collectors class providing functionality to collect the processed data into List, Setor Map through streaming.

18. How to collect sorted elements into list using stream api?

- list.stream().sorted().collect(Collectors.toList());

19. Collect the sorted elements into List through streaming and display list?

- list.stream().sorted().collect(Collectors.toList()).forEach(System.out::println);

20. What is parallel streaming?

- By default, every stream in java is sequential unless we specify the parallel stream explicitly as follows
    - Stream st = collection.stream().paralle();

21. Differentiate forEach() and forEachOrdered()?

- forEach() method is not guarantee about order of elements in parallel streaming
- forEachOrdered() method is guarantee about elements order.

22. How to display the elements greater than 5 in a List using filter() method?

- list.stream().filter(n->n>5).forEach(System.out::println);

23. How to collect the elements greater than 5 in List using filter() method?

- List<Integer> res = list.stream().filter(n->n>5).collect(Collectors.toList());

24. What is Optional Class?

- Optional class is used to handle NullPointerException

25. What is Java Predicate?

- It is a functional interface belongs to java.util.function package.
- It predicates the input argument and returns a Boolean value.

```
@FunctionalInterface
interface Predicate{
        public boolean test(E e);
}
```

26. How can we create Predicate that display only integers greater than 5?

- Predicate<E> pr = n -> n>5;

27. How to filter list elements which are greater than 5 using predicate?

- Predicate<Integer> pr = n -> n>5;
- list.stream().filter(pr).forEach(System.out::println);

28. How to filter list of strings starts with 'A' using predicate and parallel stream and please guarantee about the order of elements?

- Predicate<String> pr = s->s.startsWith("A");
- List.stream().parallel().filter(pr).forEachOrdered(System.out::println);

29. How can we pass a method reference to predicate?

- Predicate<E> pr = ClassName::methodName;
    - Or
- Predicate<E> pr = new ClassName::methodName;

Please complete this worksheet

Which of these packages contain all the collection classes?

Java.lang	Java.util
Java.net	Java.awt

Which of these classes is not part of Java's collection framework?

Maps	Array
Stack	Queue

Which of this interface is not a part of Java's collection framework?

List	Set
SortedSet	SortedList

What is Collection in Java?

A group of Objects	A group of classes
A group of interfaces	None of the above

Which interface restricts duplicate elements?

Set	List
Queue	All of these

What implementation of Iterator can traverse a collection in both directions?

Iterator	ListIterator
SetIterator	MapIterator

Which does NOT extends the Collection interface?

List	Map
Set	None of the above

Which provides better performance for the insertion and removal from the middle of the list?

Vector	ArrayList
LinkedList	All of these

The collection

extends Collections class	extends Iterable interface
implements Serializable interface	implements Traversable interface

List, Set and Queue_____Collection.

Extends	Implements
both of the above	none of the above

Genie Ashwani

Which among the following Sets maintains insertion order?

HashSet	TreeSet
LinkedHashSet	Both b and c

Which among the following stores element in ascending order?

ArrayList	HashSet
TreeSet	Both A and C

Which interface provides the capability to store objects using a key-value pair?

Java.util.Map	Java.util.Set
Java.util.List	Java.util.Collection

Which collection class allows you to associate its elements with key values, and allows you to retrieve objects in FIFO (first-in, first-out) sequence?

java.util.ArrayList	java.util.LinkedHashMap
java.util.HashMap	java.util.TreeMap

List, Set and Queue_____Collection.

Inherit	Implement
Both A and B	None of the above

Which class stores elements in ascending order?

ArrayList	HashSet
TreeSet	All of the above

Elements of which of the collection can be traversed using Enumeration?

ArrayList	Vector
HashSet	TreeMap

How to point last element in the ArrayList?

list.length	list.length – 1
list.size()	list.size() – 1

How to access first element from the ArrayList?

list[0]	list[1]
list.first()	list.get(0)

Arraylist, Linkedlist and Vector are _____

Interfaces	Enums
Classes	None of the above

Genie Ashwani

List, Set and Queue_____Collection.

extends	implements
Both A and B are true	None of the above

Which class stores elements in ascending order?

ArrayList	HashSet
TreeSet	All the answers are true

How to point last element in the ArrayList?

list.length	list.length – 1
list.size()	list.size() - 1

How to access first element from the ArrayList?

list[0]	list[1]
list.get(0)	list.get(1)

How to set second value in ArrayList as 5?

list[1] = 5;	list.set(2, 5);
list.set(5, 1);	list.set(1, 5);

How to remove the value 3 from the list [5, 3, 2, 1]?

list.remove(3);	list.remove(0);
list.remove(1);	list.remove(2);

How to add 2 between the 1 and 3 in ArrayList [1, 3, 4]?

list.add(2, 0);	list.add(2, 1);
list.add(0, 2);	list.add(1, 2);

Which is correct syntax to access an element of an ArrayList in Java?

list.get(index)	list[index]
list.getElement(index)	list.index

How do you get the number of elements in an ArrayList called "list" in Java?

list.size();	list.count();
list.length();	list.getSize();

What is the list nums if it is initially [5, 3, 1] and the following code is executed?

```
nums.add(6);
nums.add(0, 4);
nums.remove(1)
;
```

[4, 3, 1, 6]	[5, 3, 1, 6]

[4, 3, 6]	[4, 5, 3, 6]

Given the nums ArrayList with the values [5, 4, 3, 2], what statement below removes the value 3 from the list?

nums.remove(2)	nums.remove(3)
nums.get(2) = null	nums.remove(1)

Which of these is not a interface in the Collections Framework?

Collection	Group
Set	List

What implementation of Iterator can traverse a collection in both directions?

Iterator	ListIterator
SetIterator	MapIterator

The Comparable interface contains which called?

toCompare	compare
compareTo	compareWith

What is the default number of Partitions/segments in Concurrent Hash Map?

12	32
4	16

Which is best suited to a multi-threaded environment?

WeakHashMap	Hashtable
HashMap	ConcurrentHashMap

The default capacity of a Vector is:

10	12
8	16

Which does NOT extends the Collection interface?

List	Map
Set	Queue

The default capacity of a ArrayList is:

12	16
1	10

Genie Ashwani

Which provides better performance for the insertion and removal from the middle of the list?

Vector	ArrayList
LinkedList	(All of these)

After resizing, size of ArrayList is increased by :

200%	50%
100%	(None of these)

After resizing, size of Vector is increased by:

200%	100%
50%	(None of these)

What guarantees type-safety in a collection?

Generics	Abstract classes
Interfaces	Collection

Which of these is synchronized?

ArrayList	LinkedList
Vector	(None of these)

Which does not allow to store a null value?

TreeSet	LinkedHashSet
HashSet	None

Which of these is synchronized?

TreeMap	HashMap
Hashtable	All

Which of these class should be preferred to be used as a key in a HashMap?

String	Integer
Double	Any of these

What should we use when add and remove operations are more frequent than get operations?

LinkedList	ArrayList
Vector	All

Genie Ashwani

Consider the list = [13, 22, 96, 23, 13, 76, 22, 45]

1. Display all values of list

2. Display elements of list in reverse order

3. Display even count and odd count in the list

4. Display only first even number in the given list

5. Display last odd number in the given list

Genie Ashwani

6. Find the sum of all elements in the list

7. Find the sum of elements in the list which are divisible by 2 but not with 3.

8. Find the sum of even numbers and odd number of given list

9. Find the biggest element in the given list

10. Find the smallest element in the given list

Genie Ashwani

Consider the ArrayList of Employee objects(records)

id	name	salary	deptNum	location	age
101	amar	30000	20	Hyderabad	34
102	harin	35000	10	Chennai	45
103	Sathya	40000	20	Bangalore	23
104	Annie	45000	20	Hyderabad	32
105	Raji	42000	30	Pune	19
106	Vijji	50000	10	Bangalore	27

Display details of all records using for-each loop

Display List in reverse order.

Display List using Iterator

Display employee details whose ID is 105

Genie Ashwani

Display Employee details in reverse order using ListIterator

Display employee details belongs to Hyderbad location

Display employee details belongs to DeptNum 20 and location is Bangalore

Display employee details whose salary in between 30000 to 40000

Display employee details with Ids 102 and 105

Genie Ashwani

Display employee details belongs to Hyderabad and salary is greater than 35000

Display all employee details except Hyderabad location.

Display employee details belongs to deptNum 10 and not belongs to Chennai

Check any employee belongs to Hyderabad location.

Display employee details with minimum age

Genie Ashwani

Implement programs to following problems

1. Program to display the size of list.
2. Program to check the list is empty or not.
3. Program to add element to list at particular location.
4. Program to check the list is empty or not without isEmpty() method.
5. Write a method to remove duplicates from a list.
6. Given two lists, write a function to merge them into a single sorted list.
7. Write a function to find the average of elements in a list.
8. Write a program to sort a list of strings in alphabetical order.
9. Create a method to count the occurrences of a specific element in a list.
10. Write a function to split a list into sublists of a given size.
11. Implement a function to remove a specific element from a set.
12. Write a program to convert a set to an array.
13. Write a Java program to find the size of a map.
14. Implement a method to check if a map is empty.
15. Create a function to get the value associated with a specific key in a map.
16. Write a program to merge two maps into a single map.
17. Implement a method to remove a key-value pair from a map.
18. Create a program to find the keys of the highest value in a map.
19. Create a method to get all the values from a map as a list.
20. Write a Java program to find the common elements between two lists.
21. Implement a method to find the difference between two sets.
22. Create a function to check if a map contains a specific value.
23. Write a program to combine two lists into a single list without duplicates.
24. Implement a method to remove all elements from a list that are present in a set.
25. Write a program to check if a map contains any null values.
26. Implement a method to find the index of the first occurrence of an element in a list efficiently.
27. Create a function to find the common elements between multiple sets efficiently.
28. Write a program to efficiently find the median of a list.
29. Create a method to efficiently find the frequency of each element in a list.
30. Implement a function to efficiently remove all occurrences of a specific element from a list.

MCQs on JDK8 features

Which of the following is a new feature introduced in JDK 8?

Enhanced for loop	Generics
Lambda expressions	Abstract classes

What is the primary use of lambda expressions in Java 8?

To simplify exception handling	To create new objects
To enable functional programming	To create multiple threads

What is the purpose of the default keyword in interfaces in JDK 8?

- To specify default values for variables
- To mark a method as abstract
- To allow multiple inheritance in Java
- To provide a default implementation for interface methods

Which method is used to convert a stream of elements to an array in Java 8?

toArray()	convert()
asArray()	streamToArray()

Which interface is used to represent a sequence of elements and supports operations in Java 8?

Iterator	List
Stream	Sequence

Method reference to an instance method of an arbitrary object of a particular type look like?

ClassName::methodName	instance::methodName
methodName::instance	methodName::ClassName

What does the filter() method do in the Stream API?

- It sorts the elements in the stream.
- It removes duplicate elements from the stream.
- It selects elements based on a given predicate and returns a new stream.
- It performs an action on each element of the stream.

In Java 8, which interface represents a supplier of results and does not take any argument?

Function	Runnable
Consumer	Supplier

Which of the following is NOT a new Date and Time API class introduced in JDK 8?

LocalDate	LocalTime
DateTime	LocalDateTime

What is the purpose of the forEach() method in the Stream API?

- To filter elements based on a predicate
- To map elements to a different type
- To perform an action on each element of the stream
- To find the maximum element in the stream

What is the purpose of the reduce() method in the Stream API?

- To combine the elements of the stream into a collection
- To convert the stream to an array
- To perform a reduction operation on the elements of the stream and return a single value
- To sort the elements of the stream

What is the purpose of the Collectors class in the Stream API?

- To provide utility methods for creating streams.
- To perform mutable reduction operations on streams and collect the results.
- To sort the elements of a stream.
- To filter the elements of a stream based on a predicate.

Which method is used to sort the elements of a stream using a custom comparator in Jdk8?

sort()	sorted()
order()	arrange()

What is the purpose of the Predicate functional interface in Java 8?

- To represent a supplier of results.
- To represent a function that takes one argument and returns a boolean.
- To represent a consumer of data.
- To represent an operation that accepts three input arguments and returns no result.

Which method is used to combine multiple predicates into a single predicate in Java 8?

test()	and()
combine()	merge()

Which of the following is NOT a terminal operation in the Stream API?

forEach()	map()
collect()	reduce()

Genie Ashwani

Which method is used to calculate the sum of elements in a stream of integers in Java 8?

collect()	sum()
reduce()	aggregate()

In the context of parallel streams, how can you ensure the order of elements in the output?

- Use the unordered() method before the terminal operation.
- Use the forEachOrdered() method after the terminal operation.
- Use the sequential() method before the terminal operation.
- Use the parallel() method before the terminal operation.

What is the purpose of the Consumer functional interface in Java?

- To represent a supplier of results.
- To represent a function that takes one argument and returns a value.
- To represent a function that takes one argument and returns no result.
- To represent a function that takes no arguments and returns a result.

Which method is used to perform an action on each element of an Iterable in Java?

forEach()	forLoop()
process()	loop()

What is the return type of the map() method in the Stream API?

List	Set
Stream	Map

Which method is used to combine the elements of a stream into a single value in Java?

collect()	reduce()
combine()	aggregate()

What is the purpose of the Supplier functional interface in Java?

- To represent a supplier of results.
- To represent a function that takes one argument and returns a value.
- To represent a function that takes one argument and returns no result.
- To represent a function that does not take any arguments and returns a result.

Which method is used to remove elements from a stream based on a given predicate?

remove()	delete()
filter()	exclude()

What is the return type of the collect() method in the Stream API when used to collect elements into a List?

Set	Map
List	Collection

In the context of Streams, what is the purpose of the forEach() method?

- To perform an action on each element of the stream.
- To collect the elements of the stream into a collection.
- To combine the elements of the stream into a single value.
- To sort the elements of the stream.

Which method is used to apply a function to each element of a stream and collect the results in a collection?

forEach()	map()
collect()	process()

Which method is used to perform an action on each element of the stream and return void?

forEach()	collect()
map()	reduce()

What is the purpose of the distinct() method in the Stream API?

- To sort the elements of the stream.
- To remove duplicates from the stream.
- To filter the elements based on a predicate.
- To combine the elements of the stream into a single value.

In the context of parallel programming with streams, how can you ensure the order of elements in the output?

- By using the parallel() method before the terminal operation.
- By using the sequential() method before the terminal operation.
- By using the unordered() method before the terminal operation.
- By using the forEachOrdered() method after the terminal operation.

Genie Ashwani