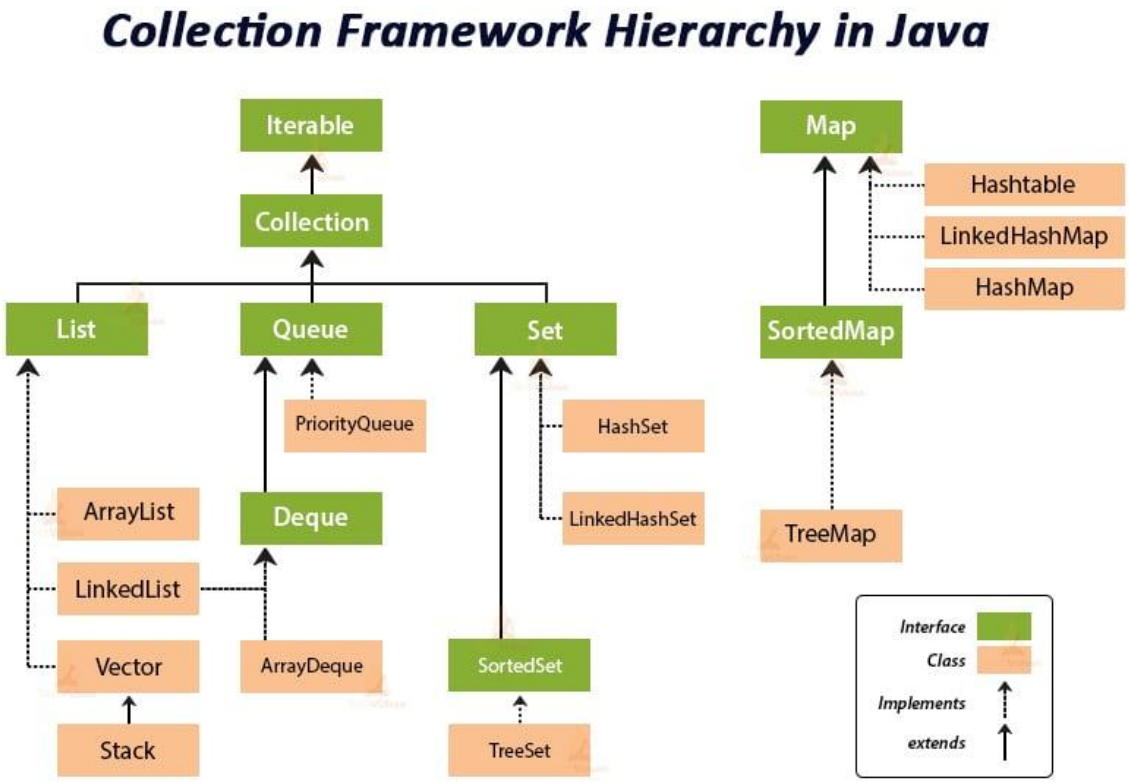
Java Collections Interview Questions



1) What is the Collection framework in Java?

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

---

2) What are the main differences between array and collection?

Array and Collection are somewhat similar regarding storing the references of objects and manipulating the data, but they differ in many ways. The main differences between the array and Collection are defined below:

- Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
- Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
- Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

3) Explain various interfaces used in Collection framework?

Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework. List of interfaces of Collection Framework is given below:

1. Collection interface: Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.

Syntax:

```
1. public interface Collection<E>extends Iterable
```

Where <E> represents that this interface is of Generic type

2. List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

Syntax:

```
1. public interface List<E> extends Collection<E>
```

3. Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface

Syntax:

```
1. public interface Set<E> extends Collection<E>
```

Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).

Syntax:

```
1. public interface Queue<E> extends Collection<E>
```

4. Dequeue interface: it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.

Syntax:

```
1. public interface Dequeue<E> extends Queue<E>
```

5. Map interface: A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

---

4) What is the difference between ArrayList and Vector?

No.	ArrayList	Vector
1)	ArrayList is not synchronized.(not thread safe)	Vector is synchronized.(thread safe)
2)	ArrayList is not a legacy class.	Vector is a legacy class.
3)	ArrayList increases its size by 50% of the array size.	Vector increases its size by doubling the array size.
4)	ArrayList is not ?thread-safe? as it is not synchronized.	Vector list is ?thread-safe? as it?s every method is synchronized.

---

5) What is the difference between ArrayList and LinkedList?

No.	ArrayList	LinkedList
1)	ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.

2)	ArrayList is not efficient for manipulation because too much is required.	LinkedList is efficient for manipulation.
3)	ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
4)	ArrayList provides random access.	LinkedList does not provide random access.
5)	ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

6) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in the forward direction only whereas ListIterator traverses the elements into forward and backward direction.

No.	Iterator	ListIterator
1)	The Iterator traverses the elements in the forward direction only.	ListIterator traverses the elements in backward and forward directions both.
2)	The Iterator can be used in List, Set, and Queue.	ListIterator can be used in List only.
3)	The Iterator can only perform remove operation while traversing the collection.	ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection.

7) What is the difference between Iterator and Enumeration?

No.	Iterator	Enumeration
1)	The Iterator can traverse legacy and non-legacy elements.	Enumeration can traverse only legacy elements.

2)	The Iterator is fail-fast.	Enumeration is not fail-fast.
3)	The Iterator is slower than Enumeration.	Enumeration is faster than Iterator.
4)	The Iterator can perform remove operation while traversing the collection.	The Enumeration can perform only traverse operation on the collection.

8) What is the difference between List and Set?

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

- The List can contain duplicate elements whereas Set includes unique items.
- The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
- The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
- The List interface can allow n number of null values whereas Set interface only allows a single null value.

9) What is the difference between HashSet and TreeSet?

The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below.

- HashSet maintains no order whereas TreeSet maintains ascending order.
- HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
- HashSet performs faster than TreeSet.
- HashSet is backed by HashMap whereas TreeSet is backed by TreeMap.

10) What is the difference between Set and Map?

The differences between the Set and Map are given below.

- Set contains values only whereas Map contains key and values both.

- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

11) What is the difference between HashSet and HashMap?

The differences between the HashSet and HashMap are listed below.

- HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
- HashSet implements Set interface whereas HashMap implements the Map interface
- HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
- HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

12) What is the difference between HashMap and TreeMap?

The differences between the HashMap and TreeMap are given below.

- HashMap maintains no order, but TreeMap maintains ascending order.
- HashMap is implemented by hash table whereas TreeMap is implemented by a Tree structure.
- HashMap can be sorted by Key or value whereas TreeMap can be sorted by Key.
- HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

13) What is the difference between HashMap and Hashtable?

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

But there are many differences between HashMap and Hashtable classes that are given below.

HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

14) What is the difference between Collection and Collections?

The differences between the Collection and Collections are given below.

- The Collection is an interface whereas Collections is a class.
- The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
- The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

15) What is the difference between Comparable and Comparator?

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.**	**The Comparator provides** multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.**
2) Comparable affects the original class**, i.e., the actual class is modified.**	**Comparator** doesn't affect the original class**, i.e., the actual class is not modified.**
3) Comparable provides compareTo() method **to sort elements.**	**Comparator provides** compare() method **to sort elements.**
4) Comparable is present in java.lang **package.**	**A Comparator is present in the** java.util **package.**
5) We can sort the list elements of Comparable type by Collections.sort(List) **method.**	**We can sort the list elements of Comparator type by** Collections.sort(List, Comparator) **method.**

Java Comparable Example

16) What do you understand by BlockingQueue?

BlockingQueue is an interface which extends the Queue interface. It provides concurrency in the operations like retrieval, insertion, deletion. While retrieval of any element, it waits for the queue to be non-empty. While storing the elements, it waits for the available space.

BlockingQueue cannot contain null elements, and implementation of BlockingQueue is thread-safe.

Syntax:

```
1. public interface BlockingQueue<E> extends Queue <E>
```

17) What is the advantage of Properties file?

If you change the value in the properties file, you don't need to recompile the java class. So, it makes the application easy to manage. It is used to store information which is to be changed frequently. Consider the following example.

```
1. import java.util.*;
2. import java.io.*;
3. public class Test {
4. public static void main(String[] args)throws Exception{
5.     FileReader reader=new FileReader("db.properties");
6.
7.     Properties p=new Properties();
8.     p.load(reader);
9.
10.    System.out.println(p.getProperty("user"));
11.    System.out.println(p.getProperty("password"));
12.}
13.}
```

Output

```
system
oracle
```

18) What does the hashCode() method?

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number if two keys (by calling equals() method) are identical.

However, it is possible that two hash code numbers can have different or the same keys.

If two objects do not produce an equal result by using the equals() method, then the hashcode() method will provide the different integer result for both the objects.

19) Why we override equals() method?

The equals method is used to check whether two objects are the same or not. It needs to be overridden if we want to check the objects based on the property.

For example, Employee is a class that has 3 data members: id, name, and salary. However, we want to check the equality of employee object by the salary. Then, we need to override the equals() method.

20) How to synchronize List, Set and Map elements?

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

public static List synchronizedList(List l){}
public static Set synchronizedSet(Set s){}
public static SortedSet synchronizedSortedSet(SortedSet s){}
public static Map synchronizedMap(Map m){}
public static SortedMap synchronizedSortedMap(SortedMap m){}

21) What is the advantage of the generic collection?

There are three main advantages of using the generic collection.

- If we use the generic class, we don't need typecasting.
- It is type-safe and checked at compile time.
- Generic confirms the stability of the code by making it bug detectable at compile time.

22) What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value are known as hash-collision. Two separate entries will be kept in a single hash bucket to avoid the collision. There are two ways to avoid hash-collision.

- Separate Chaining
- Open Addressing

---

23) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

---

24) What is the default size of load factor in hashing based collection?

The default size of load factor is **0.75**. The default capacity is computed as initial capacity * load factor. For example, 16 * 0.75 = 12. So, 12 is the default capacity of Map.

---

25) What do you understand by fail-fast?

The Iterator in java which immediately throws ConcurrentmodificationException, if any structural modification occurs in, is called as a Fail-fast iterator. Fail-fats iterator does not require any extra space in memory.

---

26) What is the difference between Array and ArrayList?

The main differences between the Array and ArrayList are given below.

SN	Array	ArrayList
1	The Array is of fixed size, means we cannot resize the array as per need.	ArrayList is not of the fixed size we can change the size dynamically.

2	Arrays are of the static type.	ArrayList is of dynamic size.
3	Arrays can store primitive data types as well as objects.	ArrayList cannot store the primitive data types it can only store the objects.

27) What is the difference between the length of an Array and size of ArrayList?

The length of an array can be obtained using the property of length whereas ArrayList does not support length property, but we can use size() method to get the number of objects in the list.

Finding the length of the array

```
1. Int [] array = new int[4];
2. System.out.println("The size of the array is " + array.length);
3.
```

Finding the size of the ArrayList

```
1. ArrayList<String> list=new ArrayList<String>();
2. list.add("ankit");
3. list.add("nippun");
4. System.out.println(list.size());
5.
```

28) How to convert ArrayList to Array and Array to ArrayList?

Array to ArrayList:-

We can convert an Array to ArrayList by using the asList() method of Arrays class. asList() method is the static method of Arrays class and accepts the List object. Consider the following syntax:

```
1. Arrays.asList(item)
2. Exa:- String[] myArray = new String[]{"Python","Java"};
3. ArrayList<String> list = new ArrayList<String>(Arrays.asList(myArray));
```

ArrayList to an Array:-

We can convert an ArrayList to Array using toArray() method of the ArrayList class. Consider the following syntax to convert the ArrayList to the List object.

1. List_object.toArray(**new** String[List_object.size()])
2. Exa:- String[] myArray = list.toArray(new String[list.size()]);

---

29) How to make Java ArrayList Read-Only?

We can obtain java ArrayList Read-only by calling the Collections.unmodifiableCollection() method. When we define an ArrayList as Read-only then we cannot perform any modification in the collection through add(), remove() or set() method.

---

30) How to remove duplicates from ArrayList?

There are two ways to remove duplicates from the ArrayList.

- **Using HashSet:** By using HashSet we can remove the duplicate element from the ArrayList, but it will not then preserve the insertion order.
- **Using LinkedHashSet:** We can also maintain the insertion order by using LinkedHashSet instead of HashSet.

The Process to remove duplicate elements from ArrayList using the LinkedHashSet:

- Copy all the elements of ArrayList to LinkedHashSet.
- Empty the ArrayList using clear() method, which will remove all the elements from the list.
- Now copy all the elements of LinkedHashset to ArrayList.

---

31) How to reverse ArrayList?

To reverse an ArrayList, we can use reverse() method of Collections class. Consider the following example.

1. import java.util.ArrayList;
2. import java.util.Collection;
3. import java.util.Collections;
4. import java.util.Iterator;

```
5. import java.util.List;
6. public class ReverseArrayList {
7. public static void main(String[] args) {
8.     List list = new ArrayList<>();
9.     list.add(10);
10.    list.add(50);
11.    list.add(30);
12.    Iterator i = list.iterator();
13.    System.out.println("printing the list....");
14.    while(i.hasNext())
15.    {
16.        System.out.println(i.next());
17.    }
18.    Iterator i2 = list.iterator();
19.    Collections.reverse(list);
20.    System.out.println("printing list in reverse order....");
21.    while(i2.hasNext())
22.    {
23.        System.out.println(i2.next());
24.    }
25.  }
26.}
```

Output

```
printing the list....
10
50
30
printing list in reverse order....
30
50
10
```

32) How to sort ArrayList in descending order?

To sort the ArrayList in descending order, we can use the reverseOrder method of Collections class. Consider the following example.

```
1. import java.util.ArrayList;
2. import java.util.Collection;
```

```
3. import java.util.Collections;
4. import java.util.Comparator;
5. import java.util.Iterator;
6. import java.util.List;
7.
8. public class ReverseArrayList {
9. public static void main(String[] args) {
10.     List list = new ArrayList<>();
11.     list.add(10);
12.     list.add(50);
13.     list.add(30);
14.     list.add(60);
15.     list.add(20);
16.     list.add(90);
17.
18.     Iterator i = list.iterator();
19.     System.out.println("printing the list....");
20.     while(i.hasNext())
21.     {
22.         System.out.println(i.next());
23.     }
24.
25.    Comparator cmp = Collections.reverseOrder();
26.    Collections.sort(list,cmp);
27.     System.out.println("printing list in descending order....");
28.     Iterator i2 = list.iterator();
29.     while(i2.hasNext())
30.     {
31.         System.out.println(i2.next());
32.     }
33.
34. }
35. }
```

Output

```
printing the list....
10
50
```

```
30
60
20
90
printing list in descending order....
90
60
50
30
20
10
```

33) How to synchronize ArrayList?

We can synchronize ArrayList in two ways.

- Using Collections.synchronizedList() method
- Using CopyOnWriteArrayList<T>

34) When to use ArrayList and LinkedList?

LinkedLists are better to use for the update operations whereas ArrayLists are better to use for the search operations.

35) What is the Difference between List, Set, and Map in Java?

List	Set	Map
The list interface allows duplicate elements. Any number of duplicate elements can be inserted into the list without affecting the same existing values and their indexes.	Set does not allow duplicate elements. Set and all of the classes which implements Set interface should have unique elements.	The Map stored the elements as key & value pair. Map doesn't allow duplicate keys while it allows duplicate values.

List	Set	Map
List and all of its implementation classes maintains the insertion order.	Set does not maintain any insertion order. Still few of its classes sort the elements in an order such as LinkedHashSet maintains the element in insertion order.	The map also does not maintain any insertion order. However few of its classes does the same. i.e. TreeMap sorts the map in the ascending order of keys and Linked HashMap sorts the element in the insertion order, the order in which the elements got added to the LinkedHashMap
We can add any number of null values.	But in set almost only one null value.	The map allows a single null key at most and any number of null values.
List implementation classes are Array List, LinkedList.	Set implementation classes are HashSet, LinkedHashSet, and TreeSet.	Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap.
The list provides get() method to get the element at a specified index.	Set does not provide get method to get the elements at a specified index	The map does not provide get method to get the elements at a specified index
If you need to access the elements frequently by using the index then we can use the list	If you want to create a collection of unique elements then we can use set	If you want to store the data in the form of key/value pair then we can use the map.
To traverse the list elements by using Listlterator.	Iterator can be used traverse the set elements	Through keyset, value, and entry set.
for(String s:list){ System.out.println(s);	for(String s:hSet){ System.out.println(s);	for(Entry<Integer,String> me:hmap.entrySet()){

List	Set	Map
}	}	System.out.println("Key"+me.getKey()+" & Value"+me.getValue()); }

➔ **2d Array in collection is same as Map.**

36) Iterate, sort and reversed the list using different ways?

```
package com.collection.list;

public class Employee1 {

	private Integer id;
	private String firstName;
	private String lastName;
	private Integer age;

	public Employee1(Integer id, String firstName, String lastName, Integer
age) {
		this.id = id;
		this.firstName = firstName;
		this.lastName = lastName;
		this.age = age;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
```

```
	public Integer getAge() {
		return age;
	}

	public void setAge(Integer age) {
		this.age = age;
	}

	@Override
	public String toString() {
		return "\n[" + this.id + "," + this.firstName + "," +
this.lastName + "," + this.age + "]";
	}

}
package com.collection.list;

public class Employee implements Comparable<Employee>{

	private Integer id;
    private String firstName;
    private String lastName;
    private Integer age;

    public Employee(Integer id, String firstName, String lastName, Integer
age){
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }



    public Integer getId() {
		return id;
	}



	public void setId(Integer id) {
		this.id = id;
	}



	public String getFirstName() {
		return firstName;
	}



	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}



	public String getLastName() {
```

```
		return lastName;
	}



	public void setLastName(String lastName) {
		this.lastName = lastName;
	}



	public Integer getAge() {
		return age;
	}



	public void setAge(Integer age) {
		this.age = age;
	}



	@Override
    public String toString() {
       return
"\n["+this.id+","+this.firstName+","+this.lastName+","+this.age+"]";
    }



	@Override
	public int compareTo(Employee o) {
		// TODO Auto-generated method stub
		return this.id - o.id;
	}
}
```

```
package com.collection.list;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;
import java.util.Vector;

public class ListExample {

	public static void main(String[] args) {

		Stack<String> colour = new Stack<>(); //Vector<>();//
LinkedList<>(); //ArrayList<>();
		colour.add("Red");
```

```
		colour.add("Green");
		colour.add("Blue");
		colour.add("Green");
		colour.add("Yellow");
		colour.add("");

		System.out.println(colour);

		System.out.println("----------------------------Simple For Loop--
------------------------------");


		for(int i=0;i<colour.size();i++)
		{
			System.out.println(colour.get(i));
		}

		System.out.println("----------------------------ForEach Loop-----
----------------------------");

		for(String s: colour)
		{
			System.out.println(s);
		}

		System.out.println("----------------------------Iterator and
while loop---------------------------------");

		Iterator<String> iterator = colour.iterator();
		while (iterator.hasNext()) {
		   System.out.println(iterator.next());
		}

		System.out.println("----------------------------While loop-------
-------------------------");

		int num=0;
		while(colour.size() > num)
		{
			System.out.println(colour.get(num));
			num++;
		}

		System.out.println("----------------------------Lambda
Expression---------------------------------");

		   colour.forEach(colorsName -> System.out.println(colorsName));

		System.out.println("----------------------------Method Reference-
-------------------------------");

		colour.forEach(System.out::println);

		System.out.println("----------------------------Sorting of List--
------------------------------");

		Collections.sort(colour);
		colour.forEach(System.out::println);
```

```
            System.out.println("----------------------------Sorting in
Descending order---------------------------------");

            Collections.sort(colour, Collections.reverseOrder());
            colour.forEach(System.out::println);

            System.out.println("----------------------------Sorting of
objects - Comparable interface---------------------------------");

            Stack<Employee> employees  = new Stack<>();  //ArrayList<>();
                employees.add(new Employee(6,"Yash", "Chopra", 25));
                employees.add(new Employee(2,"Aman", "Sharma", 28));
                employees.add(new Employee(3,"Aakash", "Yaadav", 52));
                employees.add(new Employee(5,"David", "Kameron", 19));
                employees.add(new Employee(4,"James", "Hedge", 72));
                employees.add(new Employee(8,"Balaji", "Subbu", 88));
                employees.add(new Employee(7,"Karan", "Johar", 59));
                employees.add(new Employee(1,"Lokesh", "Gupta", 32));
                employees.add(new Employee(9,"Vishu", "Bissi", 33));
                employees.add(new Employee(10,"Lokesh", "Ramachandran", 60));

             // UnSorted List
              System.out.println(employees);

              Collections.sort(employees);

              System.out.println("After Sorting ");
              // Default Sorting by employee id
              System.out.println(employees);

              System.out.println("----------------------------Sorting of
objects - Comparator interface---------------------------------");

              Stack<Employee1> employees1  = new Stack<>(); //ArrayList<>();
                employees1.add(new Employee1(6,"Yash", "Chopra", 25));
                employees1.add(new Employee1(2,"Aman", "Sharma", 28));
                employees1.add(new Employee1(3,"Aakash", "Yaadav", 52));
                employees1.add(new Employee1(5,"David", "Kameron", 19));
                employees1.add(new Employee1(4,"James", "Hedge", 72));
                employees1.add(new Employee1(8,"Balaji", "Subbu", 88));
                employees1.add(new Employee1(7,"Karan", "Johar", 59));
                employees1.add(new Employee1(1,"Lokesh", "Gupta", 32));
                employees1.add(new Employee1(9,"Vishu", "Bissi", 33));
                employees1.add(new Employee1(10,"Lokesh", "Ramachandran",
60));

                // UnSorted List
              System.out.println(employees1);

              //Sort all employees by id
              System.out.println("Sort by id");
              employees1.sort(Comparator.comparing(e -> e.getId()));
//employees1.sort(Comparator.comparing(Employee1::getId));
              System.out.println(employees1);


              //Sort all employees by first name
              System.out.println("Sort by first name");
```

```
                employees1.sort(Comparator.comparing(e -> e.getFirstName()));
//employees1.sort(Comparator.comparing(Employee1::getFirstName));
                System.out.println(employees1);


              //Sort all employees by last name
                System.out.println("Sort by last name");
                employees1.sort(Comparator.comparing(e -> e.getLastName()));
//employees1.sort(Comparator.comparing(Employee1::getLastName));
                System.out.println(employees1);

              //Sort all employees by age
                System.out.println("Sort by age");
                employees1.sort(Comparator.comparing(e -> e.getAge()));
//employees1.sort(Comparator.comparing(Employee1::getAge));
                System.out.println(employees1);


                System.out.println("--------Sort by firstName in reverse order--
------");
                //Sort all employees by first name; And then reversed
                Comparator<Employee1> comparator = Comparator.comparing(e ->
e.getFirstName());
                employees1.sort(comparator.reversed());

                //Let's print the sorted list
                System.out.println(employees1);

                System.out.println("----------------------------Sorting of
multiple objects - Comparator interface---------------------------------");

              //Sorting on multiple fields; Group by.
                Comparator<Employee1> groupByComparator =
Comparator.comparing(Employee1::getFirstName)

.thenComparing(Employee1::getLastName);
                employees1.sort(groupByComparator);

                System.out.println(employees1);
        }
}
```

37)What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

Q38) Different ways to iterate a list.

Solution:-

1. Using the traditional for loop with an index:

```
            List<String> list = Arrays.asList("A", "B", "C");
```

```
for (int i = 0; i < list.size(); i++) {
    String item = list.get(i);
    System.out.println(item);
}
```

2. Using the Enhanced for Loop (for-each loop):

```
List<String> list = Arrays.asList("A", "B", "C");

for (String item : list) {
    System.out.println(item);
}
```

3. Using the Iterator:

```
List<String> list = Arrays.asList("A", "B", "C");
Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {
      String item = iterator.next();
      System.out.println(item);

}
```

4. Using a while loop:

```
List<String> list = Arrays.asList("A", "B", "C");
int num = 0;

while (list.size()>num) {
      System.out.println(list.get(num));
      num++;

}
```

5. Using Java 8 Streams and Lambdas:

```
List<String> list = Arrays.asList("A", "B", "C");


list.forEach(item -> System.out.println(item));
```

6. Using forEach loop in Java 8 with Method Reference:

```
List<String> list = Arrays.asList("A", "B", "C");

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

Q39) Different ways to iterate a set.

Solution:-

1. Using the traditional for loop with an index:

```
Set<String> set = new HashSet<>();
set.add("hello");
set.add("world");
set.add("!");

for (int i = 0; i < set.size(); i++) {
    String item = set.get(i);
    System.out.println(item);
}
```

2. Using the Enhanced for Loop (for-each loop):

```
Set<String> set = new HashSet<>();
set.add("hello");
set.add("world");
set.add("!");

for (String item : set) {
    System.out.println(item);
}
```

3. Using the Iterator:

```
Set<String> set = new HashSet<>();
set.add("hello");
set.add("world");
set.add("!");
Iterator<String> iterator = set.iterator();

while (iterator.hasNext()) {
      String item = iterator.next();
      System.out.println(item);

}
```

4. Using a while loop:

```
Set<String> set = new HashSet<>();
set.add("hello");
set.add("world");
set.add("!");
int num = 0;

while (set.size()>num) {
      System.out.println(set.get(num));
      num++;
```

```
}
```

5. Using Java 8 Streams and Lambdas:

```
Set<String> set = new HashSet<>();
            set.add("hello");
            set.add("world");

            set.add("!");


            set.forEach(item -> System.out.println(item));
```

6. Using forEach loop in Java 8 with Method Reference:

```
Set<String> set = new HashSet<>();
            set.add("hello");
            set.add("world");

            set.add("!");


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

Q40) Different ways to iterate a Map.

Solution:-

1. Using the Enhanced for Loop (for-each loop):

```
Map<String, Integer> map = new HashMap<>();
map.put("hello", 1);
map.put("world", 2);
map.put("!", 3);

for (Map.Entry<String, Integer> entry : map.entrySet()) {
      String key = entry.getKey();
      Integer value = entry.getValue();

      System.out.println(key + ": " + value);
}

for (String key : map.keySet()) {
      Integer value = map.get(key);
      System.out.println(key + ": " + value);
}

for (Integer value : map.values()) {
      System.out.println(value);
}
```

3. Using the Iterator:

```
Map<String, Integer> map = new HashMap<>();
             map.put("hello", 1);
             map.put("world", 2);
             map.put("!", 3);

             Iterator<Map.Entry<String, Integer>> iterator =
map.entrySet().iterator();
             while (iterator.hasNext()) {
               Map.Entry<String, Integer> entry = iterator.next();

               String key = entry.getKey();
               Integer value = entry.getValue();

               System.out.println(key + ": " + value);

             }
```

3. Using Java 8 Streams and Lambdas and Method Reference:

```
Map<String, Integer> map = new HashMap<>();
             map.put("hello", 1);
             map.put("world", 2);
             map.put("!", 3);

             map.entrySet().stream().forEach(System.out::println);
             map.entrySet().stream().forEach(entry -> {
                 String key = entry.getKey();
                 Integer value = entry.getValue();
                 System.out.println(key + ": " + value);
             });
             map.keySet().stream().forEach(key -> {
                 Integer value = map.get(key);
                 System.out.println(key + ": " + value);
             });
             map.values().stream().forEach(value -> {
                   System.out.println(value);
             });
             map.forEach((key, value) -> {
                   System.out.println(key + ": " + value);
             });
```