**Q55) What are the concrete subclasses of Map?**

* HashMap
* LinkedHashMap
* TreeMap
* Hashtable
* WeakHashMap
* IdentityHashMap
* EnumMap

**Q56) Why Map is not extending Collection interface?**

The Map interface is not part of the Collection hierarchy because it deals with key-value pairs, whereas the Collection interface deals with single elements. The operations of Map (like adding, removing, and accessing entries) are quite different from those of a Collection (like adding, removing, or iterating over elements). Thus, Map is a separate interface.

**Q57) What is Entry in Map?**

Entry is a nested interface inside Map that represents a key-value pair. The Map.Entry interface provides methods to access the key and value of a map entry, such as getKey(), getValue(), and setValue().

**Q58) How to get all the keys of Map?**

You can retrieve all the keys of a Map using the keySet() method, which returns a Set of keys:

java

Copy code

Set<K> keys = map.keySet();

**Q59) How to get all the values of Map?**

You can retrieve all the values of a Map using the values() method, which returns a Collection of values:

java

Copy code

Collection<V> values = map.values();

**Q60) How to get all the Entry objects of Map?**

You can get all the entries (key-value pairs) of a Map using the entrySet() method, which returns a Set of Map.Entry objects:

java

Copy code

Set<Map.Entry<K, V>> entries = map.entrySet();

**Q61) What is the use of Properties class?**

The Properties class is a subclass of Hashtable and is used to maintain a list of key-value pairs, where both the keys and values are String objects. It is often used for configuration settings, such as loading and storing properties from .properties files.

**Q62) What are the differences between HashMap and Hashtable?**

* **Thread Safety**: HashMap is not synchronized (not thread-safe), while Hashtable is synchronized (thread-safe).
* **Null Values**: HashMap allows one null key and multiple null values, while Hashtable does not allow any null keys or values.
* **Performance**: HashMap is faster than Hashtable because it is not synchronized.
* **Inheritance**: HashMap extends AbstractMap, while Hashtable extends Dictionary.

**Q63) What are the differences between HashMap and TreeMap?**

* **Ordering**: HashMap does not guarantee any order of keys, while TreeMap sorts the keys based on their natural ordering or by a custom comparator.
* **Performance**: HashMap has constant-time performance (O(1)) for basic operations like get() and put(), while TreeMap has logarithmic time (O(log n)).
* **Null Keys**: HashMap allows one null key, but TreeMap does not allow null keys.

**Q64) What are the legacy classes available under Map hierarchy?**

The primary legacy class in the Map hierarchy is Hashtable. Other classes like Properties also extend from Hashtable.

**Q65) What is Generics?**

Generics allow types (classes and interfaces) to be parameterized. They enable stronger type checks at compile time, avoiding runtime ClassCastException, and allow for the creation of classes, interfaces, and methods that can operate on any type.

**Q66) Which version of Java has introduced Generics concept?**

Generics were introduced in **Java 5 (JDK 1.5)**.

**Q67) What are the advantages of using Generics in Collection?**

* **Type Safety**: Ensures compile-time type checking, avoiding runtime errors.
* **Code Reusability**: Generic methods and classes work with any type.
* **Elimination of Casts**: Avoids explicit casting and reduces the chances of ClassCastException.

**Q68) What are various syntaxes available to use Generics?**

* **Class declaration**:

java

Copy code

class Box<T> { T value; }

* **Method declaration**:

java

Copy code

public <T> void add(T element) { }

* **Generic with multiple types**:

java

Copy code

class Pair<K, V> { K key; V value; }

**Q69) What is Diamond Operator and which version of Java has introduced it?**

The Diamond Operator (<>) simplifies the use of generics by inferring the type from the context. It was introduced in **Java 7**. Example:

java

Copy code

List<String> list = new ArrayList<>();

**Q70) What is the use of Queue interface?**

The Queue interface represents a collection designed for holding elements prior to processing, typically in a **FIFO (First In First Out)** manner. It is used in scenarios such as task scheduling and buffering.

**Q71) Which version of Java has introduced Queue concept?**

The Queue interface was introduced in **Java 5**.

**Q72) What are important subclasses of Queue?**

* LinkedList
* PriorityQueue
* ArrayDeque
* ConcurrentLinkedQueue
* DelayQueue

**Q73) Can I add null value in PriorityQueue?**

No, you cannot add null values to a PriorityQueue. It throws a NullPointerException.

**Q74) How can I store custom objects in PriorityQueue?**

You need to implement the Comparable interface in the custom class or provide a Comparator to define the ordering.

java

Copy code

PriorityQueue<MyObject> queue = new PriorityQueue<>(new MyComparator());

**Q75) What are the important methods of PriorityQueue?**

* offer(E e): Adds an element to the queue.
* peek(): Retrieves but does not remove the head of the queue.
* poll(): Retrieves and removes the head of the queue.
* size(): Returns the size of the queue.

**Q76) What is use of Collections class?**

The Collections class provides static utility methods to operate on or return collections. For example, methods like sort(), reverse(), shuffle(), and unmodifiableList().

**Q77) What is the difference between Collection and Collections?**

* Collection is an interface that represents a group of objects, such as List, Set, etc.
* Collections is a utility class that provides methods to operate on collections.

**Q78) How can I convert List to Enumeration object?**

You can convert a List to an Enumeration object by using the Collections.enumeration() method, which takes a Collection as an argument.

java

Copy code

List<String> list = new ArrayList<>();

list.add("A");

list.add("B");

Enumeration<String> enumeration = Collections.enumeration(list);

**Q79) How can I convert Enumeration to List object?**

You can convert an Enumeration to a List using a loop to iterate over the elements and add them to a List.

java

Copy code

List<String> list = new ArrayList<>();

Enumeration<String> enumeration = ...;

while (enumeration.hasMoreElements()) {

list.add(enumeration.nextElement());

}

**Q80) How can I get Immutable version of List subclass object?**

You can get an immutable version of a List using Collections.unmodifiableList():

java

Copy code

List<String> list = new ArrayList<>();

List<String> immutableList = Collections.unmodifiableList(list);

**Q81) How can I get Immutable version of Set subclass object?**

You can get an immutable version of a Set using Collections.unmodifiableSet():

java

Copy code

Set<String> set = new HashSet<>();

Set<String> immutableSet = Collections.unmodifiableSet(set);

**Q82) How can I get Immutable version of Map subclass object?**

You can get an immutable version of a Map using Collections.unmodifiableMap():

java

Copy code

Map<String, String> map = new HashMap<>();

Map<String, String> immutableMap = Collections.unmodifiableMap(map);

**Q83) What is the reason for UnsupportedOperationException?**

UnsupportedOperationException is thrown when an operation is not supported by the collection. This commonly happens with immutable collections (e.g., when trying to add an element to an unmodifiable collection) or fixed-size collections that don’t support certain operations like adding/removing elements.

**Q84) How can I sort the elements of a Collection in forward order?**

You can sort the elements in forward order using Collections.sort():

java

Copy code

List<String> list = new ArrayList<>();

Collections.sort(list); // Sorts in natural ascending order

**Q85) How can I sort the elements of a Collection in reverse order?**

To sort elements in reverse order, you can use Collections.sort() along with Collections.reverseOrder():

java

Copy code

List<String> list = new ArrayList<>();

Collections.sort(list, Collections.reverseOrder());

**Q86) What is the use of Arrays class?**

The Arrays class provides utility methods for array manipulation, such as sorting, searching, copying, converting to a list, and filling elements. Some commonly used methods are sort(), binarySearch(), copyOf(), asList(), etc.

**Q87) What is the method to print contents of an array?**

You can use Arrays.toString() to print the contents of a one-dimensional array:

java

Copy code

int[] array = {1, 2, 3};

System.out.println(Arrays.toString(array));

For multidimensional arrays, you can use Arrays.deepToString():

java

Copy code

int[][] array = {{1, 2}, {3, 4}};

System.out.println(Arrays.deepToString(array));

**Q88) How can I convert a simple array to List type?**

You can convert an array to a List using Arrays.asList():

java

Copy code

String[] array = {"A", "B", "C"};

List<String> list = Arrays.asList(array);

**Q89) What is the use of copyOf() method of Arrays class?**

The copyOf() method is used to copy an array into a new array with the specified length. It can also be used to resize an array.

java

Copy code

int[] array = {1, 2, 3};

int[] newArray = Arrays.copyOf(array, 5); // Copies elements and resizes

**Q90) What is the use of Date class and why is it not recommendable to use?**

The Date class represents a specific instant in time, but it is not recommended because:

* It has many deprecated methods (like getYear(), getMonth()).
* It lacks support for time zones and locales.
* It is generally replaced by the Calendar and java.time classes introduced in Java 8, which are more powerful and flexible.

**Q91) What is the value getYear() method returns when it is called on 25-March-1988?**

The getYear() method returns the year minus 1900. So, if it is called on 25-March-1988, it will return 88.

**Q92) What is the value getMonth() method returns when it is called on 25-March-1988?**

The getMonth() method returns 0 for January, 1 for February, and so on. For 25-March-1988, it will return 2 (since March is the third month, and indices start from 0).

**Q93) What is the use of Calendar class?**

The Calendar class is used for date and time manipulation. It supports various operations like getting the current date and time, adding or subtracting time, and converting between time zones.

**Q94) What are the options available to get the Calendar instance?**

You can get a Calendar instance using the following methods:

* Calendar.getInstance() (gets the current date and time).
* Calendar.getInstance(TimeZone zone) (with a specific time zone).
* Calendar.getInstance(Locale locale) (with a specific locale).

**Q95) What is the use of SimpleDateFormat class and where is it available?**

SimpleDateFormat is used to format and parse dates in a locale-sensitive manner. It is part of the java.text package.

java

Copy code

SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");

String formattedDate = sdf.format(new Date());

**Q96) What is the use of StringTokenizer class?**

The StringTokenizer class is used to break a string into tokens based on a delimiter (default delimiter is space). It is an older utility, and String.split() or Scanner is preferred in modern Java.

**Q97) How can I access the elements from StringTokenizer object?**

You can access elements by using the hasMoreTokens() and nextToken() methods.

java

Copy code

StringTokenizer tokenizer = new StringTokenizer("Hello World");

while (tokenizer.hasMoreTokens()) {

System.out.println(tokenizer.nextToken());

}

**Q98) What is Locale class?**

The Locale class represents a specific geographical, political, or cultural region, and it is used to localize operations like formatting dates, numbers, and currencies.

**Q99) What is ResourceBundle class?**

The ResourceBundle class is used for loading localized resources, such as property files, based on the Locale. It helps support internationalization (i18n) by separating the text for different languages.

**Q100) What is properties file?**

A .properties file is a simple key-value pair file used to store configuration data, such as settings or translations for an application.

**Q101) How can I support I18N for an application?**

To support **Internationalization (I18N)** in a Java application, you need to:

1. Use Locale to handle different regions and languages.
2. Store localized content in ResourceBundle files (usually .properties files) where keys represent text, and values represent translations.
3. Format dates, numbers, and currencies using the appropriate Locale.
4. Retrieve localized content using ResourceBundle.

For example, create a messages\_en.properties for English and messages\_fr.properties for French, and load them using:

java

Copy code

Locale locale = new Locale("fr", "FR");

ResourceBundle bundle = ResourceBundle.getBundle("messages", locale);

String greeting = bundle.getString("greeting");

**Q102) How can I get the instance of ResourceBundle class?**

You can get an instance of ResourceBundle using the getBundle() method. It loads the appropriate bundle based on the Locale:

java

Copy code

Locale locale = new Locale("en", "US");

ResourceBundle bundle = ResourceBundle.getBundle("messages", locale);

**Q103) How can I access the keys from ResourceBundle instance?**

You can access all the keys in a ResourceBundle using the keySet() or getKeys() method:

java

Copy code

ResourceBundle bundle = ResourceBundle.getBundle("messages", new Locale("en", "US"));

Set<String> keys = bundle.keySet();

for (String key : keys) {

System.out.println(key + ": " + bundle.getString(key));

}

**Q104) What is the use of Timer & TimerTask?**

* Timer is a utility class that schedules tasks for execution at specific times or intervals.
* TimerTask is an abstract class that implements Runnable, representing the task to be scheduled by the Timer.

You can use these classes to schedule periodic tasks, such as sending reminders or performing background operations at specific intervals.

**Q105) How can I schedule a task for execution?**

You can schedule tasks using Timer.schedule() or scheduleAtFixedRate() methods. Example:

java

Copy code

Timer timer = new Timer();

TimerTask task = new TimerTask() {

public void run() {

System.out.println("Task executed!");

}

};

timer.schedule(task, 1000); // Schedules task to run after 1 second

**Q106) What is Scanner class?**

The Scanner class is part of java.util and is used to read input from various sources, including the console, files, or strings. It can tokenize input based on whitespace or a custom delimiter.

Example of reading input from the console:

java

Copy code

Scanner scanner = new Scanner(System.in);

System.out.println("Enter your name: ");

String name = scanner.nextLine();

System.out.println("Hello, " + name);

**Q107) How can I use a delimiter with Scanner?**

You can set a custom delimiter using the useDelimiter() method. For example, to split input based on a comma:

java

Copy code

Scanner scanner = new Scanner("A,B,C");

scanner.useDelimiter(",");

while (scanner.hasNext()) {

System.out.println(scanner.next());

}

**Q108) What is Formatter class?**

The Formatter class is used to create formatted output, similar to printf() in C. It supports formatting strings, numbers, dates, and other data types. It is commonly used with String.format() or System.out.printf() to format output.

Example:

java

Copy code

Formatter formatter = new Formatter();

formatter.format("Name: %s, Age: %d", "John", 25);

System.out.println(formatter);

formatter.close();

Formatter is also used in logging, reporting, and other scenarios where formatted output is needed.