METHODS

**1. Can we overload the main in Java?**

Yes you can overload the main method in Java, nothing wrong with this but Java will only call your specific main method, i.e. main method with the following signature:

public static void main(String[] args) or public static void main(String args...) which is the main method as variable argument method and only supported post-Java 5 world.

**2. Can we override main in Java?**

No, you can not override the main method in Java, Why? because main is a static method and in Java static method is bonded during compile time and you can not override the static method in Java. If you declare a method with the same name and signature it's called method hiding.

**3. Can we make the main final in Java?**

Of course, you can make the main method final in Java. JVM has no issue with that. Unlike any final method, you can not override main in Java.

**4. Can we make the main synchronized in Java?**

Yes, the main can be synchronized in Java, a synchronized modifier is allowed in the main signature and you can make your main method synchronized in Java.

**5. How to call a nonstatic method from main in Java?**

This question applies not only to main but all static methods in Java. Since nonstatic methods can not be called from static context directly, you need to first create an Object as a local variable and then you can call a non-static method using that object

**CONSTRUCTORS**

**6.What is a Constructor?**

Constructors are used to initialize the object’s state. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.

**7.Do we have Copy Constructor in Java?**

Like C++, Java also supports copy constructor. But, unlike C++, Java doesn’t create a default copy constructor if you don’t write your own.

To copy the values of one object into another in java, you can use:

• Constructor

• Assigning the values of one object into another

• clone() method of Object class

**8.What is Constructor Chaining ?**

Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call same class constructor where as super() is used to call super class constructor.

// Java program to illustrate Constructor Chaining

// within same class Using this() keyword

class Temp

{

// default constructor 1

// default constructor will call another constructor

// using this keyword from same class

Temp()

{

// calls constructor 2

this(5);

System.out.println("The Default constructor");

}

// parameterized constructor 2

Temp(int x)

{

// calls constructor 3

this(5, 15);

System.out.println(x);

}

// parameterized constructor 3

Temp(int x, int y)

{

System.out.println(x \* y);

}

public static void main(String args[])

{

// invokes default constructor first

new Temp();

}

}

**9.Can we call sub class constructor from super class constructor?**

No. There is no way in java to call sub class constructor from a super class constructor.

**10.What happens if you keep a return type for a constructor?**

Ideally, Constructor must not have a return type. By definition, if a method has a return type, it’s not a constructor.(JLS8.8 Declaration) It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.Example:

class GfG

{

int GfG()

{

return 0; // Warning for the return type

}

}

**11.What is No-arg constructor?**

Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

class GfG

{

public GfG()

{

//No-arg constructor

}

}

**12.How a no – argument constructor is different from default Constructor?**

If a class contains no constructor declarations, then a default constructor with no formal parameters and no throws clause is implicitly declared.

If the class being declared is the primordial class Object, then the default constructor has an empty body. Otherwise, the default constructor simply invokes the superclass constructor with no arguments.

**13.What are private constructors and where are they used?**

Like any method we can provide access specifier to the constructor. If it’s made private, then it can only be accessed inside the class.

The major scenarios where we use private constructor:

• Internal Constructor chaining

• Singleton class design pattern

**14.When do we need Constructor Overloading?**

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. Different constructors can do different work by implementing different line of codes and are called based on the type and no of parameters passed.

According to the situation , a constructor is called with specific number of parameters among overloaded constructors.

**15.Do we have destructors in Java?**

No, Because Java is a garbage collected language you cannot predict when (or even if) an object will be destroyed. Hence there is no direct equivalent of a destructor.

**16.When is a Constructor called?**

Each time an object is created using a new() keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the data members of the same class.

**17.How many types of Constructors are in Java?**

There are two types of constructors in Java:

1. Default Constructor (Non-parameterized Constructor)

2. Parameterized Constructor

**18.Is there any method to call a sub-class constructor from a superclass constructor?**

The subclass constructor has its own private data members, so Java does not provide any way to access the sub-class constructor from a super class constructor. However, we can call a superclass constructor from a sub-class constructor by using the super keyword.

**19.Can we have a constructor in the Interface?**

No, we cannot have constructors in the Java interface.

**20.what will happen when a constructor is declared as protected?**

Generally, when we declare a method as protected, other classes can access that method in a different package by using inheritance only. But, when we declare a constructor protected, it behaves slightly differently than a method. The protected constructor can only be accessed by using a super keyword according to Java language standards.

**21.Why constructor name is similar to the class name?**

When we create an object of a class using a new keyword, it should have information about that particular class. That is why the constructor's name must be similar to the class name.

**COLLECTIONS**

**22.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.

**23.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:

o 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.

o Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.

o Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

**24.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:**

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:**

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:**

public interface Set<E> extends Collection<E>

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

**Syntax:**

public interface Queue<E> extends Collection<E>

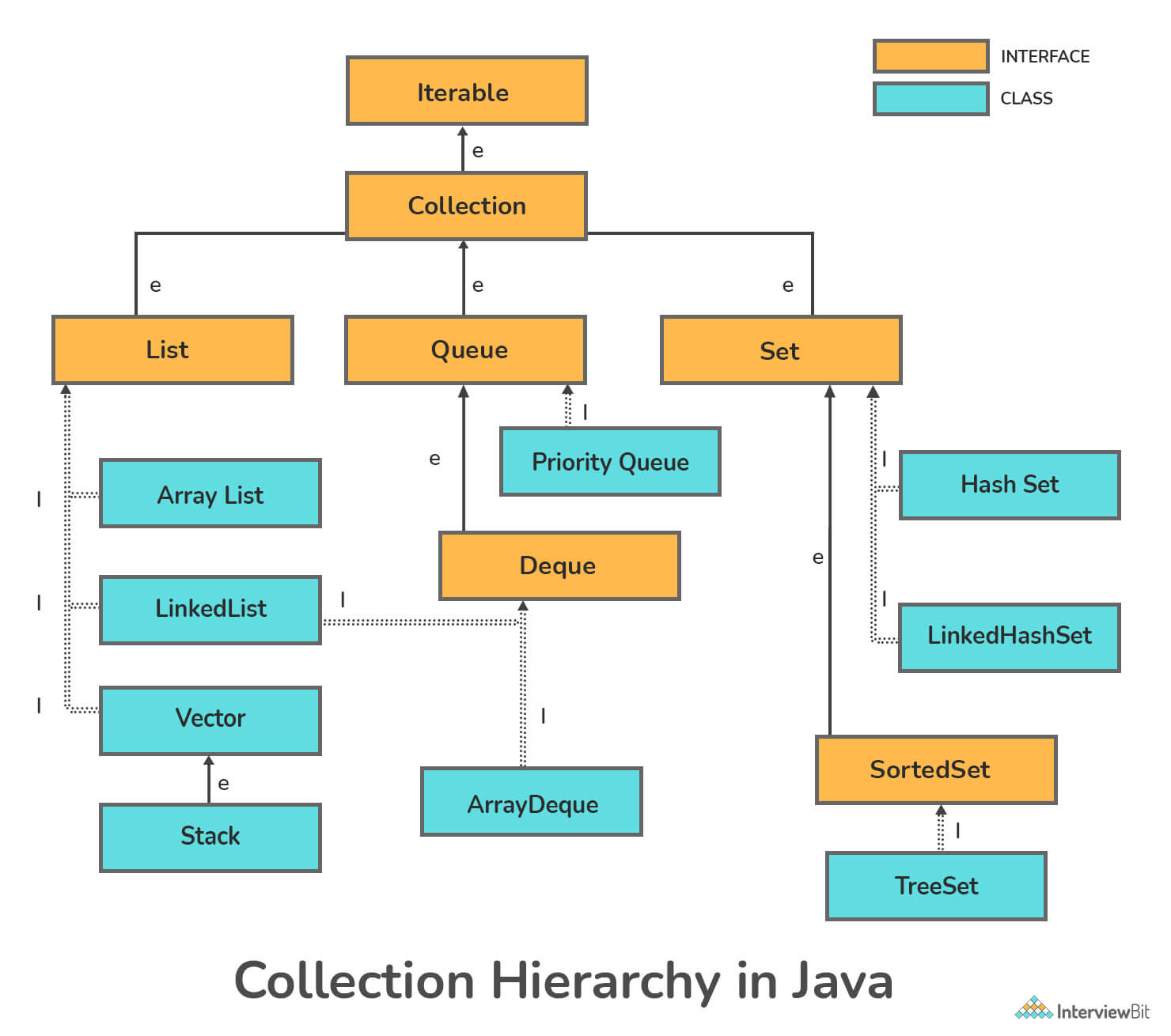
**5. Dequeue interface:** it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.

**Syntax:**

public interface Dequeue<E> extends Queue<E>

**25.Explain the hierarchy of the Collection framework in Java.**

The entire collection framework hierarchy is made up of four fundamental interfaces: Collection, List, Set, Map, and two specific interfaces for sorting called SortedSet and SortedMap. The java.util package contains all of the collection framework's interfaces and classes. The following diagram depicts the Java collection structure.

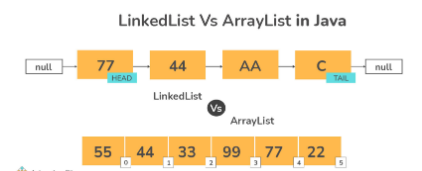


Here, **e**denotes extends, **i**denotes implements

* **Extends**: The keyword extends is used to create inheritance between two classes and two interfaces.
* **Implements**: The keyword implements are used to create inheritance across classes and interfaces.

**26.Difference between ArrayList and LinkedList.**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface |



**MAP**

**27.How does the put() method of HashMap works in Java? (**[**answer**](http://www.java67.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html)**)**

The put() method of HashMap works in the principle of hashing. It is responsible for storing an object into the backend array. The hashcode() method is used in conjunction with a hash function to find the correct location for the object into the bucket. If a collision occurs, then the entry object which contains both key and value is added to a [linked list,](http://www.java67.com/2016/01/how-to-implement-singly-linked-list-in-java-using-generics-example.html)and that linked list is stored into the bucket location.

**28.Can you store a null key in Java HashMap? (**[**answer**](http://javarevisited.blogspot.sg/2011/02/how-hashmap-works-in-java.html)**)**

Yes, HashMap allows one null key, which is stored at the first location of bucket array e.g., bucket[0] = value. The HashMap doesn't call hashCode() on the null key because it will throw [NullPointerException](http://www.java67.com/2012/09/nullpointerexception-in-java-nemesis-of.html), hence when a user call the get() method with null, then the value of the first index is returned.

**29.Which data structure HashMap represents? (**[**answer**](http://www.java67.com/2013/08/ata-structures-in-java-programming-array-linked-list-map-set-stack-queue.html)**)**

The HashMap is an implementation of hash table data structure, which is idle for mapping one value to other like id to name as you can search for value in O(1) time if you have the key.

**30.What are the different ways to iterate over HashMap in Java? (**[**answer**](https://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)**)**

Here are some of the ways to iterate over HashMap in Java:

* by using keySet and iterator
* by using entrySet and iterator
* by using entrySet and enhanced for loop
* by using keySet and get() method

**31.In which order mappings are stored in HashMap? (**[**answer**](http://www.java67.com/2012/08/difference-between-hashmap-and-LinkedHashMap-Java.html)**)**

### Random order because HashMap doesn't provide any ordering guarantee for keys, values, or entries. When you iterate over a HashMap, you may get a different order every time you iterate over it. 32.How does resize happens in HashMap? ([answer](http://www.java67.com/2012/09/top-10-tricky-java-interview-questions-answers.html))

The resizing happens when the map becomes full or when the size of the map crosses the load factor. For example, if the load factor is 0.75 and then becomes more than 75% full, then resizing trigger, which involves an array copy. First, the size of the bucket is doubled, and then old entries are copied into a new bucket.

**33.Differentiate between HashMap and TreeMap.**

|  |  |
| --- | --- |
| **HashMap** | **TreeMap** |
| Doesn’t preserves any ordering | Preserves the natural ordering |
| Implicitly implements the hashing principle | Implicitly implements the Red-Black Tree Implementation |
| Can store only one null key | Cannot store any null key |
| More memory usage | Less memory usage |
| Not synchronized | Not synchronized |

**34.Differentiate between ArrayList and Vector.**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| Non-synchronized in nature | Synchronized in nature |
| It is not a legacy class | Is a legacy class |
| Increases size by 1/2 of the ArrayList | Increases size by double of the ArrayList |
| It is not thread-safe | It is thread-safe |

**STRINGS**

## ****35.How to create a string?****

To create a String object in Java, there are two ways:

* Using the new operator. For example,

|  |  |
| --- | --- |
| 1 | String s1 = new String("Joey"); |

* Using a string literal or constant expression. For example,

|  |  |
| --- | --- |
| 1  2 | String s1="Joey"; (string literal) or  String s1="Joe" + "y"; (string constant expression) |

## ****36.What is String Pool in Java?****

String Pool is a storage area in Java heap.

String allocation, like all [object allocation](https://www.edureka.co/blog/java-object/), proves to be a costly affair in both the cases of time and memory. The JVM performs some steps while initializing string literals to increase performance and decrease memory overhead. To decrease the number of String objects created in the JVM, the String class keeps a pool of strings.

Each time a string literal is created, the [JVM](https://www.edureka.co/blog/java-architecture/#componentsofjava) checks the string literal pool first. If the string already exists in the string pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object initializes and is placed in the pool.

### 37.What is String in Java?

Strings, one of the most common objects used in Java programming, are essentially sequences of characters. As an example, the string "Scaler" contains the following characters: "S", "c", "a", "l", "e", and "r". You can either create a string by using String Literal or by using the NEW keyword.

**38.How to declare a string in Java?**

String declaration in Java can be done in two ways:

* **By string literal**: Double quotes are used to create Java String literals.
  + Example: String str= "Scaler";
* **By new keyword**: Keyword "new" is used to create a Java string.
  + Example: String str=new String ("Scaler");

### 39.Is String a primitive or derived type in Java?

Strings are derived data types. Strings are Java objects that represent sequences of characters. String objects are created using the java.lang.String class. There are many functions that need to be called upon when processing a string, such as substring(), indexof(), equals(), toUppercase(), etc, which primitives types do not have.

### 40.What does the string intern() method do in Java?

If you apply the intern() method to a few strings, you will ensure that all strings having the same content share the same memory. As soon as a String object is invoked with intern(), it first checks if the string value of the String object is already present in the string pool and if it is available, then the reference to that string from the string constant pool is returned. If not, a new string object is added to the string pool, and a reference to it is returned.

**41.State the difference between StringBuffer and StringBuilder in Java.**

StringBuffer and StringBuilder are two Java classes for manipulating strings. These are mutable objects, i.e., they can be modified, and provide various methods such as insert(), substring(), delete(), and append(), for String manipulation.

* **StringBuffer**: The StringBuffer class was created by the Java Team when they realized the need for an editable string object. Nevertheless, StringBuffer has all methods synchronized, meaning they are thread-safe. Therefore, StringBuffer allows only one thread to access a method at once, so it is not possible to call StringBuffer methods from two threads simultaneously, which means it takes more time to access. The StringBuffer class has synchronized methods, making it thread-safe, slower, and less efficient than StringBuilder. The StringBuffer class was introduced in Java 1.0.
  + Syntax:

StringBuffer var = new StringBuffer(str);

* **StringBuilder**: It was at that point that the Java Team realized that making all methods of StringBuffer synchronized wasn't the best idea, which led them to introduce StringBuilder. The StringBuilder class has no synchronized methods. Unlike StringBuffer, StringBuilder does not offer synchronized methods, which makes it less thread-safe, faster, and more efficient. StringBuilder was introduced in Java 1.5 in response to StringBuffer's shortcomings.
  + Syntax:

StringBuilder var = new StringBuilder(str);

### 42. Is it possible to compare Strings using the == operator? If so, what is the risk involved?

Yes, you can compare strings using the == operator. One can use == operators for reference comparison (address comparison). The majority of the time, developers compare strings with the == operator, instead of using the equals() method, resulting in an error.

**ARRAYS**

### 43.On which memory arrays are created in Java?

Arrays are created on dynamic memory by JVM. There is no question of static memory in Java everything (variable, array, object, etc.) is created on dynamic memory only.

### 44.Can we call the main() method of a class from another class?

Yes! We can call the main() method of a class from another class using Classname.main(). At the time of calling the main() method, we should pass a string type array to it.

### 45.What is an array in Java?

An array is a finite and ordered collection of homogeneous data elements. It is finite because it contains a limited number of elements. It is ordered because all the elements are stored one by one in a contiguous location of computer memory (heap) in a linear fashion. It is homogeneous because all elements of an array are of the same data type only. We can store either primitive types or object references into it.

**46.What are the types of an array?**

Arrays are generally categorized into two parts as described below:

* Single Dimensional Array
* Multi-Dimensional Array (2D and 3D arrays)

### 47.Is it possible to declare array size as negative?

No, it is not possible to declare array size as negative. Still, if we declare the negative size, there will be no compile-time error. But we get the NegativeArraySizeException at run-time.

**48.What is the difference between int array[] and int[] array?**

There is no difference between array[] and []array. Both array[] and []array are the ways to declare an array. The only difference between them is that if we are declaring more than one array in a line, we should use prefix []. If we are declaring a single array in a line, we should use postfix []. For example, consider the following declaration:

1. **int** array1[], array2;   //array1[] is an array while array2 is just a variable of type int
2. **int**[] arr1, arr2;  //both arr1 and arr2 are arrays of int type

**49.What is the default value of the array?**

When we create a new array, it always initialized with the default values. The default values of the array are:

* If an array is of byte, short, int, and long type, the default value is **0**.
* If an array is of float and double type, the default value is **0**.
* If an array is of Boolean type, the default value is **false**.
* If an array is of an Object type, the default value is **null**.

### 50.What do you understand by the jagged array?

A jagged array is a multidimensional array in which member arrays are of different sizes. For example, int array[][]=new int[3][]. The statement creates a two-dimensional jagged array

**51.How many ways to find the duplicate elements in an array?**

There are the following five ways to find the duplicate array in Java.

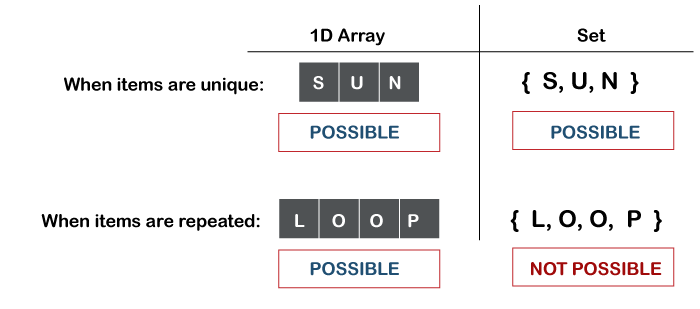
* **Brute Force Method:** In this method, we compare each element of an array with the other elements. If any of the two elements are found equal, we consider them as duplicates. The method has time complexity O(n2).
* **Using HashSet:** We can also use the HashSet class to find the duplicate elements in an array. To find the duplicate elements, iterate over the array elements and insert them into HashSet by invoking add() method of the HashSet class. If the method returns false it means that the element is already present in the Set. It takes O(n) time to find the duplicate elements.
* **Using HashMap:** We know that HashMap uses key-value pair to store an element. When we use HashMap to find the duplicate array, we store the elements of the array as keys and the frequency of the elements as values. If the value of any key is greater than 1, the key is a duplicate element. Its time and space complexity is O(n). Using this method, we can also find the number of occurrences of duplicates.

### 52.Which operations can be performed on an array?

On an array, we can perform the searching, sorting, traversal, deletion, and insertion operation.

**53.Can a Set be an array?**

In Java, a Set is an array, but an array is not necessarily a Set. Because repetition is allowed in array but in Set. For example, consider the following figure:



### 54.Is it possible to make an array volatile?

Yes, we can make an array volatile in Java. But we only make the variable that is pointing to array volatile. If an array is changed by replacing individual elements that happen before the guarantee provided by volatile variables will not hold.

### 55.What is the difference between Array and ArrayList?

**Array:** Array is static. It is of fixed size. Its size cannot be changed once it is declared. It contains both primitive data types and objects of a class. Array does not have generic features.

**ArrayList:** ArrayList is dynamic in size. Its size or capacity automatically grows when we add element into it. It contains only the object entries. It has a generic feature.

**56.what are the advantages and disadvantages of an array?**

**Advantages of Array**

* We can store multiple elements of the same type under a single variable.
* We can implement other data structures such as Stack, Queue, Tree, etc. with the help of an array.
* We can fetch data elements using the index at run time.

**Disadvantages of Array**

* Before using an array, it is mandatory to declare its size.
* It is a static structure, so we cannot increase or decreases memory allocation.
* Insertion and deletion operations are difficult because elements stored at the contiguous memory location. It also increases the cost.
* Allocate more memory than required is the wastage of memory.

**REGULAR EXPRESSIONS**

**57.How to escape text for regular expression?**

In general, regular expression uses "\" to escape constructs, but it is painful to precede the backslash with another backslash for the Java string to compile. There is another way for users to pass string Literals to the **Pattern**, like "$5". Instead of writing \\$5 or [$]5, we can type

|  |
| --- |
| Pattern.quote("$5"); |

**58.What are the classes in Java that helps to deal with regular expressions?**

Java has a dedicated package named java.util.regex that has three classes which help to deal with regular expressions. Following is a brief description about them.

* *Pattern* – represents compiled representation of a regex. You can get a new instance by using the static ‘compile’ method which accepts a regular expression as the first argument.
* *PatternSyntaxException* – unchecked exception that occurs when there is a problem with the regular expression pattern’s syntax.
* *Matcher* – engine that interprets the pattern and does match operations for an input string. You can get a new instance by using a *Pattern* object’s matcher method.

### 59.What is a metacharacter? How is it different from an ordinary character?

A metacharacter is a character that has a special meaning to a regular expression engine. This will not be counted as a regular character by the regex engine. Examples of metacharacters are ^, $, ., \*, +, etc.

**60.Is Java Regex Thread Safe?**

Instances of Pattern class are immutable and are safe for use by multiple concurrent threads. Instances of the Matcher class are not thread safe.

**61.Explain the difference between String.matches and Matcher.matches.**

Matcher.matches() is good in terms of performance as a Matcher is created on a precompiled regular expressions, while the String.matches() recompiles the regular expression every time it executes.