**Interview Questions**

**Array**

1. **What is an anonymous array?**

* Anonymous array is an array without reference. For example,

**Example: -** Public class A{

Public static void main(String[] args)

{

System.out.println(new int[]{1, 2, 3, 4, 5}.length); //Output : 5

System.out.println(new int[]{1, 2, 3, 4, 5}[1]); //Output : 2

} }

1. **Can you pass the negative number as an array size?**

* No. You can’t pass the negative integer as an array size.
* If you pass, there will be no compile time error but you will get NegativeArraySizeException at run time.

1. **What is jagged array?**

* Jagged Arrays are multidimentional arrays having columns(member arrays) of different length.
* Jagged arrays are also known as ragged arrays. They are the arrays containing arrays of different length.

1. **Can you change the size of the array once you define it? OR Can you insert or delete the elements after creating an array?**

* No. You can’t change the size of the array once you define it.
* You can not insert or delete the elements after creating an array.
* Only you can do is change the value of the elements.

1. **There are two array objects of int type. one is containing 100 elements and another one is containing 10 elements. Can you assign array of 100 elements to an array of 10 elements?**

* Yes, you can assign array of 100 elements to an array of 10 elements provided they should be of same type. While assigning, compiler checks only type of the array not the size

**String**

1. **What is SCP(String constant pool)?**

String Pool, also known as SCP (String Constant Pool), is a special storage space in Java heap memory that is used to store unique string objects. Whenever a string object is created, it first checks whether the String object with the same string value is already present in the String pool or not, and if it is available, then the reference to the string object from the string pool is returned. Otherwise, the new string object is added to the string pool, and the respective reference will be returned.

1. **How many objects are created when we create a String object using new keyword?**

Two objects are created when a string object is created with new keyword. One will be stored in heap memory and another one is stored in SCP.

1. **SCP is present in method area till 1.6 version after that it uses Heap area. Why?**

Because SCP is a fixed size in the method area but in the heap memory, SCP can Be expandable. Therefore, the string constant pool has been moved to heap area for memory utilization only.

1. **Why Strings are Immutable?**

String objects are cached in string poll. Cached string literals are shared between Multiple so there is always a risk where one action would affect all other.

Example: if we change any value of a common field like school name, it will change for all other students. So the strings are made immutable.

1. **Why String class is final?**

String class has got special features which is not available to other java classes and making the string class final prevents subclasses to inherit.

1. **Difference Between “==” and equals() method in String?**The == operator can be used for comparing references (addresses) and the .equals() method can be used to compare content. To put it simply, == checks if the objects point to the same memory location, whereas equals() compares the values of the objects**.**
2. **Difference between String, StringBuffer and StringBuilder?**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **String** | **StringBuffer** | **StringBuilder** |
| **Storage area** | String Constant Pool (SCP) | Heap Area | Heap Area |
| **Mutability** | Immutable | Mutable | Mutable |
| **Performance** | Slow | Faster than String | Fastest as compare to StringBuffer |
| **Thread Safe** | Not Thread Safe | Thread Safe and all methods are synchronized | Not Thread Safe and methods are not synchronized |
| **Memory** | Consumes More Memory | Consumes less Memory | Consumes Less Memory |

**Object Oriented Programming (Oops)**

1. **Can the static methods be overloaded?**

Yes! There can be two or more static methods in a class with the same name but differing input parameters.

1. **Can the static methods be overridden?**

No! Declaration of static methods having the same signature can be done in the subclass

but run time polymorphism cannot take place in such cases.

Overriding or dynamic polymorphism occurs during the runtime, but the static methods

are loaded and looked up at the compile time statically. Hence, these methods cant be

overridden

1. **Difference Between Abstract Class and Interface?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Abstract keyword used to make class abstract | Interface it is a bunch of rule and guideline. |
| An abstract class extends single class only and can implements multiple interfaces. | An interface can extends at a time multiple interface. |
| Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also |
| Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| Abstract class can have final, non-final, static and non-static variables | Interface has only static and final variables |
| Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class |
| A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| **Example**: public abstract class Shape  {  public abstract void draw();  } | **Example**: public interface Drawable  {  void draw();  } |

1. **What is Marker Interface?**

It is also called “tag” interface. Interface which has blank body i.e called “marker” or “tag” interface. It is used for is decision making purpose. JVM will take decision through marker interface. There are several marker interfaces such as Serializable, Cloneable, Remote, etc

1. **Define Copy constructor in java.**

Copy Constructor is the constructor used when we want to initialize the value to the new object from the old object of the same class. Here we are initializing the new object value from the old object value in the constructor. Although, this can also be achieved with the help of object cloning.

1. **Is nested interface achievable in java?**

Yes, an interface, i.e., declared within another interface or class, is known as a nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred to by the outer interface or class. It can't be accessed directly.

Example:

interface interface\_name{

...

interface nested\_interface\_name{

...

}

}

1. Do final, finally and finalize keywords have the same function?

All three keywords have their own utility while programming.

**Final**: If any restriction is required for classes, variables, or methods, the final keyword comes in handy. Inheritance of a final class and overriding of a final method is restricted by the use of the final keyword. The variable value becomes fixed after incorporating the final keyword.

Example:

final int a=100;

a = 0; // error

The second statement will throw an error.

**Finally**: It is the block present in a program where all the codes written inside it get executed irrespective of handling of exceptions.

Example:

try {

int variable = 5;

}

catch (Exception exception) {

System.out.println("Exception occurred");

}

finally {

System.out.println("Execution of finally block");

}

**Finalize**: Prior to the garbage collection of an object, the finalize method is called so that the clean-up activity is implemented.

Example:

public static void main(String[] args) {

String example = new String("InterviewBit");

example = null;

System.gc(); // Garbage collector called

}

public void finalize() {

// Finalize called

}

1. **What is the access modifier present java?**

There are 4 access modifiers present in java.

**public** :- It can access anywhere.

**private** :- It can access within a class only.

**default** :- It can access within a package only

**protected** :- It can access within a package and outside the package of subclass. It can not applied on the class.

**private < default < protected < public**

**Exceptions Handling**

1. **What is Exception?**

* In our application, there are chances to occur abnormal condition, because of that abnormal condition normal flow of program gets disturbed or sometime program will be terminated abnormally, such situation is called as an exception
* All types of exceptions only occur at runtime.
* So, avoiding such abnormal termination we need to handle exception.
* Exception will occur because of wrong user input, file mismatch, logical mistake.

There are two types of exception

o Check Exception (Complier Time Exception)

o Uncheck Exception (Runtime Exception)

1. **What is Exception Handling?**

**a**. To avoid occurring exception we need to provide alternative way to execute rest of program, this is called as exception handling.

**b**. Keywords to handle exception: try, catch, finally, throw, throws.

**try** : The code or set of statements that may raise exception should be try block.

**catch** : This block catches the exceptions thrown in the try block.

**finally** : This block of code is always executed whether an exception has occurred in the try block or not except in one scenario explained in below question.

**throw** : Its used to throw exception at runtime. Its used in custom exception.

**throws** : Its used to declare that could be throw while execution program.

1. **Difference between Checked and Unchecked Exception?**

|  |  |
| --- | --- |
| **Checked Exception** | **Unchecked Exception** |
| Exceptions that **compiler forces** user to write handling code before compilation, it is called checked exception. | Exceptions that **compiler** **doesn’t forces** user to write handling code are called  checked exception. |
| Compulsory need to handle them. | Not compulsion to handle them. |
| It **increases** code. | It **reduces** code. |
| Eg. IOException, SqlException, FileNotFoundException, etc. | All Runtime Exception as well as all Errors are example of unchecked exception. |

1. **Difference between throw & throws?**

|  |  |
| --- | --- |
| **throw** | **throws** |
| It is used to occur custom exception. | It is used to give a chance to caller method to handle the exception. |
| This keyword is used while creating new instance of exception class. | This keyword is used with method name to declare the exception that need to propagate. |
| Can be used only with single exception at a time. | Can be used with multiple exceptions at a time. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions |

1. **What is ClassCastException in Java?**

ClassCastException is a RunTimeException which occurs when JVM unable to cast an object of one type to another type.

1. **Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?**

No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but cannot be overridden with checked exceptions.

1. **Can we write only try block without catch and finally blocks?**

No, It shows compilation error. The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.

**Collection**

1. **What is Collection?**

* Collection is an **Interface**.
* Collection Means **group of object**.
* Collection represents a **single unit of objects as a group**.
* Collection provides operations that you perform on a data such as **searching**, **sorting**, **insertion** and **deletion** on the group of objects.
* Collection Size is **Growable** in nature.
* Collection it can hold both **Homogeneous and Heterogeneous** Element.
* The Collection is a framework that provides an **architecture to** **store and manipulate** the group of objects.
* Collection Framework is a **grouping of classes and interfaces** that is used to store and manage the objects.
* It provides various **classes** like **Vector, ArrayList, HashSet, Stack**, etc. Java Collection framework can also be used for **interfaces** like **Queue, Set, List**, etc.

1. **How HashMap works?**

When we create HashMap object, HashMap instance as per **default capacity 16** buckets is created.

When we perform add **(put())** operation, it accepts data in key & value format.

Internally hashing technique is used, that **generates hashcode** for key and also **calculate index** to find bucket location for **inserting** data in HashMap instance.

It will store element at that location as a **node** format.

|  |  |  |  |
| --- | --- | --- | --- |
| prev. node addr. | Key | Value | next node addr |

Now when we perform **retrieval** (get ()) operation, it **asks for key**.

Again hashing technique is used and bucket location is identified, then equals () method is used to compare key content and if it returns true then value is retrieved.

1. **Differentiate between an Array and an ArrayList.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Array** | |  | **ArrayList** |
| An Array is a collection of simi data types stored in contiguous memory locations. | |  | An ArrayList is a class of Java Collections framework. |
| It is Fixed in Size. | |  | It is Dynamic in Size. |
| It can hold only **Homogeneous** Data  Elements. | |  | It can hold both **Homogeneous** and  **Heterogeneous** Elements. |
| A for **loop or for each loop** is used to **iterate** over an array. | |  | An **iterator** is used to **iterate** over ArrayList. |
| It is **fast** in comparison to  ArrayList as it has a fixed size. | |  | ArrayList is **Slower** than Array. |
| It is strongly typed | |  | It is loosely types |
| Array is multi-dimensional. |  |  | ArrayList is Single-dimensional |
| No need to box and unbox the elements | |  | **Needs to box** and unbox the elements |

1. **Differentiate between ArrayList and LinkedList.**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| **Seaching** Operation is **fast** compare to LinkedList | **Seaching** Operation is **slow** compare to ArrayList |
| **Get()** performance is **fast** compare to LinkedList because of ArrayList maintain Index base system. | **Get()** performance is **Slow** compare to ArrayList because of LinkedList implements **Doubly** LinkedList. |
| **Deleting** is **Slow** compare to LinkedList. | **Deleting** is **fast** compare to ArrayList. |
| ArrayList need to **shifted** to fill the space created by remove operation. | LinkedList pointer location in two neighbor **node** of the node which is going to be remove. |
| Add() is **high time complexity** for insertion. | Add() is **low time complexity** for insertion. |
| Memory **consumption** is **low** in ArrayList because of indexing. | Memory **consumption** is **High** in LinkedList because of maintain data and two pointers for neighbor node. |
| Implements **dynamic** array internally to store elements | Implements **doubly** linked list internally to store elements |
| Manipulation of elements is **slower** | Manipulation of elements is **faster** |
| ArrayList act only as a **List** | LinkedList act as a **List** and a **Queue** |

1. **Differentiate between List and Set.**

|  |  |
| --- | --- |
| **List** | **Set** |
| An **ordered** collection of elements | An **unordered** collection of elements |
| **Preserves** the insertion order | **Doesn’t preserves** the insertion order |
| **Duplicate** values are allowed | Duplicate values are **not allowed** |
| Any number of **null** values can be **stored** | **Only one null** values can be stored |
| **ListIterator** can be used to traverse the List in **any direction** | ListIterator **cannot be used** to traverse a Set |
| Contains a **legacy** class called vector | **Doesn’t** contains any legacy class |

1. **Differentiate between List and Map.**

|  |  |
| --- | --- |
| **List** | **Map** |
| Belongs to java.**util** package | Belongs to java.**util** package |
| **Extends** the Collection interface | **Doesn’t** extend the Collection interface |
| **Duplicate** elements are allowed | Duplicate keys are **not** allowed but duplicate values are |
| **Multiple** null values can be stored | Only one null key can be stored but **multiple** null values are allowed |
| Preserves the **insertion** order | **Doesn’t** maintain any insertion order |
| Stores elements based on **Array Data Structure** | Stores data in **key-value pairs** using various hashing techniques |

1. **Difference between Comparable & Comparator?**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| This interface is from java.**lang** package. | This interface is from java.**util** package. |
| It is used for **Default** sorting. | It is used for **Custom** sorting. |
| It has only one method i.e **compareTo**. | It has two methods i.e **compare** & **equals**. |
| **Programmer** decides how sorting is to be done. | **User** decides how sorting is to be done. |
| Modifies the actual class | Doesn’t modifies the actual class |

1. **Difference between Hashmap & Hashtable?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| It is **not** Synchronised. | It is Synchronised. |
| It allows **multiple** threads at a time. | It allows **single** thread at a time. |
| It is **not** thread safe. | It is thread safe. |
| **Null** key (once) & Null value is **allowed**. | **Null** key & Null value is **not** allowed. |
| Its performance is **fast**. | Its performance is **slow**. |
| Inherits **AbstractMap** class | Inherits **Dictionary** class |
| Can be traversed by **Iterator** | Can be traversed by **Iterator** and **Enumeration** |

1. **Differentiate between Iterator and ListIterator.**

|  |  |
| --- | --- |
| **Iterator** | **ListIterator** |
| Can only perform **remove** operations on the Collection elements | Can perform **add**, **remove** and **replace** operations the Collection elements |
| Can traverse List, Sets and maps | Can traverse **only Lists** |
| Can traverse the Collection in **forward** direction | Can traverse the collection in **any** direction |
| Provides **no method** to **retrieve** the index of the element | Provides **methods** to **retrieve** the index of the elements |
| iterator() method is available for the **entire** Collection Framework | listIterator() is **only** available for the collections implementing the **List** interface |

**Junit Testing**

**What is Junit Testing?**

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code. It is used to check the correctness of the software.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

**What is Mokito?**

Mockito is a Java-based mocking framework used for unit testing of Java application. Mockito plays a crucial role in developing testable applications.

The main purpose of using the Mockito framework is to simplify the development of a test by mocking external dependencies and use them in the test code. As a result, it provides a simpler test code that is easier to read, understand, and modify.

**What is @Mock?**

Using this annotation it will create the mock objects. Mock objects act as a dummy or clone of the real object in testing. They are generally created by an open-source library or a mocking framework like Mockito, EasyMock, etc. Mock objects are typically used for **behavior verification**.

**What is Stub?**

Stub objects hold predefined data and provide it to answer the calls during testing. They are referred to as a dummy object with a minimum number of methods required for a test. It also provides methods to verify other methods used to access the internal state of a stub, when necessary. Stub object is generally used for state verification.

**What is @Spy?**

A Spy is like a partial mock, which will track the interactions with the object like a mock. Additionally, it allows us to call all the normal methods of the object. Whenever we call a method of the spy object, the real method will be invoked. It will directly call the real objects without changing its behaviour.