**Features of Java :**

There are the following features in Java Programming Language.

* **Simple:** Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.
* **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.
* **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.
* **Platform Independent:** Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.
* **Secured:** Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.
* **Robust:** Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.
* **Architecture Neutral:** Java is architectural neutral as it is not dependent on the architecture. In C, the size of data types may vary according to the architecture (32 bit or 64 bit) which doesn't exist in Java.
* **Interpreted:** Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.
* **High Performance:** Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++).
* **Multithreaded:** We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.
* **Distributed:** Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
* **Dynamic:** Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

**What is the difference between JDK, JRE, and JVM?**

### JVM

JVM is an acronym for Java Virtual Machine; it is an abstract machine which provides the runtime environment in which Java bytecode can be executed. It is a specification which specifies the working of Java Virtual Machine. Its implementation has been provided by Oracle and other companies. Its implementation is known as JRE.

JVMs are available for many hardware and software platforms (so JVM is platform dependent). It is a runtime instance which is created when we run the Java class. There are three notions of the JVM: specification, implementation, and instance.

### JRE

JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

### JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

* Standard Edition Java Platform
* Enterprise Edition Java Platform
* Micro Edition Java Platform

### How many types of memory areas are allocated by JVM?

Many types:

1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects
3. **Stack:** Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

### What are the various access specifiers in Java?

In Java, access specifiers are the keywords which are used to define the access scope of the method, class, or a variable. In Java, there are four access specifiers given below.

* **Public** The classes, methods, or variables which are defined as public, can be accessed by any class or method.
* **Protected** Protected can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.
* **Default** Default are accessible within the package only. By default, all the classes, methods, and variables are of default scope.
* **Private** The private class, methods, or variables defined as private can be accessed within the class only.

### What is the purpose of static methods and variables?

The methods or variables defined as static are shared among all the objects of the class. The static is the part of the class and not of the object. The static variables are stored in the class area, and we do not need to create the object to access such variables. Therefore, static is used in the case, where we need to define variables or methods which are common to all the objects of the class.

For example, In the class simulating the collection of the students in a college, the name of the college is the common attribute to all the students. Therefore, the college name will be defined as **static**.

### What is an object?

### The Object is the real-time entity having some state and behavior. In Java, Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the ****new**** keyword.

### What is the constructor?

### The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

### How many types of constructors are used in Java?

Based on the parameters passed in the constructors, there are two types of constructors in Java.

* **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
* **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.

### What are the differences between the constructors and methods?

### 

**Collection :**

**Differentiate between Collection and Collections.**

|  |  |
| --- | --- |
| **Collection** | **Collections** |
| java.util.Collection is an interface | java.util.Collections is a class |
| Is used to represent a group of objects as a single entity | It is used to define various utility method for collection objects |
| It is the root interface of the Collection framework | It is a utility class |
| It is used to derive the data structures of the Collection framework | It contains various static methods which help in data structure manipulation |

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

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| java.util.Array is a class | java.util.ArrayList is a class |
| It is strongly typed | It is loosely types |
| Cannot be dynamically resized | Can be dynamically resized |
| No need to box and unbox the elements | Needs to box and unbox the elements |

**Differentiate between Iterable and Iterator.**

|  |  |
| --- | --- |
| **Iterable** | **Iterator** |
| Iterable is an interface | Iterator is an interface |
| Belongs to java.lang package | Belongs to java.util package |
| Provides one single abstract method called iterator() | Provides two abstract methods called hasNext() and next() |
| It is a representation of a series of elements that can be traversed | It represents the object with iteration state |

**Differentiate between ArrayList and LinkedList.**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 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 |
| Can act only as a List | Can act as a List and a Queue |
| Effective for data storage and access | Effective for data manipulation |

**. Differentiate between Comparable and Comparator.**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Present in java.lang package | Present in java.util package |
| Elements are sorted based on natural ordering | Elements are sorted based on user-customized ordering |
| Provides a single method called compareTo() | Provides to methods equals() and compare() |
| Modifies the actual class | Doesn’t modifies the actual class |

**. 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 |

**Differentiate between Set and Map.**

|  |  |
| --- | --- |
| **Set** | **Map** |
| Belongs to java.util package | Belongs to java.util package |
| Extends the Collection interface | Doesn’t extend the Collection interface |
| Duplicate values are not allowed | Duplicate keys are not allowed but duplicate values are |
| Only one null values can be stored | Only one null key can be stored but multiple null values are allowed |
| Doesn’t maintain any insertion order | Doesn’t maintain any insertion order |

**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 |

**Differentiate between Queue and Stack.**

|  |  |
| --- | --- |
| **Queue** | **Stack** |
| Based on FIFO (First-In-First-Out) principle | Based on LIFO (Last-In-First-Out) principle |
| Insertion and deletion takes place from two opposite ends | Insertion and deletion takes place the same end |
| Element insertion is called enqueue | Element insertion is called push |
| Element deletion is called dequeue | Element deletion is called pop |
| Two pointers are maintained one point to the first element and the other one points the last element on the list | Only one pointer is maintained which points to the top element on the stack |

**. Differentiate between PriorityQueue and TreeSet.**

|  |  |
| --- | --- |
| **PriorityQueue** | **TreeSet** |
| It is a type of Queue | It is based on a Set data structure |
| Allows duplicate elements | Doesn’t allows duplicate elements |
| Stores the elements based on an additional factor called priority | Stores the elements in a sorted order |

**Differentiate between the Singly Linked List and Doubly Linked List.**

|  |  |
| --- | --- |
| **Singly Linked List(SLL)** | **Doubly Linked List(DLL)** |
| Contains nodes with a data field and a next node-link field | Contains nodes with a data field, a previous link field, and a next link field |
| Can be traversed using the next node-link field only | Can be traversed using the previous node-link or the next node-link |
| Occupies less memory space | Occupies more memory space |
| Less efficient in providing access to the elements | More efficient in providing access to the elements |

**Differentiate between Iterator and Enumeration.**

|  |  |
| --- | --- |
| **Iterator** | **Enumeration** |
| Collection element can be removed while traversing it | Can only traverse through the Collection |
| Used to traverse most of the classes of the Java Collection framework | Used to traverse the legacy classes such as Vector, HashTable, etc |
| Is fail-fast in nature | Is fail-safe in nature |
| Is safe and secure | Is not safe and secure |
| Provides methods like hasNext(), next() and remove() | Provides methods like hasMoreElements() and nextElement() |

**. Differentiate between HashMap and HashTable.**

|  |  |
| --- | --- |
| **HashMap** | **HashTable** |
| It is non-synchronized in nature | It is synchronized in nature |
| Allows only one null key but multiple null values | Doesn’t allow any null key or value |
| Has faster processing | has slower processing |
| Can be traversed by Iterator | Can be traversed by Iterator and Enumeration |
| Inherits AbstractMap class | Inherits Dictionary class |

**Differentiate between HashSet and HashMap.**

|  |  |
| --- | --- |
| **HashSet** | **HasMap** |
| Based on Set implementation | Based on Map implementation |
| Doesn’t allow any duplicate elements | Doesn’t allow any duplicate keys but duplicate values are allowed |
| Allows only a single null value | Allows only one null key but any number of null values |
| Has slower processing time | Has faster processing time |
| Uses HashMap as an underlying data structure | Uses various hashing techniques for data manipulation |

**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 |

**. Differentiate between HashSet and TreeSet.**

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| Uses HasMap to store elements | Uses Treemap to store elements |
| It is unordered in nature | By default, it stores elements in their natural ordering |
| Has faster processing time | Has slower processing time |
| Uses hasCode() and equals() for comparing | Uses compare() and compareTo() for comparing |
| Allows only one null element | Doesn’t allow any null element |
| Takes up less memory space | Takes up more memory space |

**. Differentiate between Queue and Deque.**

|  |  |
| --- | --- |
| **Queue** | **Deque** |
| Refers to single-ended queue | Refers to double-ended queue |
| Elements can be added or removed from only one end | Elements can be added and removed from either end |
| Less versatile | More versatile |

**. 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 |

**. 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 |

**Differentiate between failfast and failsafe.**

|  |  |
| --- | --- |
| **failfast** | **failsafe** |
| Doesn’t allow modifications of a collection while iterating | Allows modifications of a collection while iterating |
| Throws ConcurrentModificationException | Don’t throw any exceptions |
| Uses the original collection to traverse over the elements | Uses a copy of the original collection to traverse over the elements |
| Don’t require extra memory | Require extra memory |
| **What is collections framework ?**  A framework is set of classes and interfaces to build a functionality. Java collections framework provides set of interfaces and classes for storing and manipulating collections. Collection framework contains classes and interfaces in java.util package and java.util.concurrent packages. Advantages or benefits of Collections framework : 1) High performance 2) Using this framework we can create different types of collections 3) We can create our own collection and we can extend a collection. 4) Reduces programming effort. 5) Increases speed and quality : Collections framework provides high performance, implementations of useful data structures and algorithms.  **What is collection ?**  A collection is a container which holds group of objects. Collection provides a way to manage objects easily. Collections manages group of objects as single unit. Examples include list of strings, integers etc. Here are few basic operations we do on collections : 1) Adding objects to collection. 2) Removing or deleting objects from collection. 3) Retrieving object from collection. 4) Iterating collection. 200) Difference between collection, Collection and Collections in java? collection : represent group of objects where objects are stored. Collection : This is one of the core interface which provides basic functionality for collection. Collections : Collections contains some utility static methods that operate on collections.  **Explain about Collection interface in java ?**  Collection is the fundamental and root interface in Collections framework. Collection extends Iterable interface and inherits iterator method which returns Iterator object. Signature : public interface Collection extends Iterable { } Methods in Collection interface : boolean add(E e); Adds an element to the collection. Returns true if element is added. boolean remove(Object o); Removes an object from collection if that object is present in collection. Return true if matching object is removed from collection. boolean addAll(Collection c); Adds all the elements specified in the collection to this collection.Returns true if all elements are added. boolean removeAll(Collection c); Removes all the elements from this collection that are specified in other collection.Returns true if all the elements are removed. int size(); Returns number of elements in collection. boolean isEmpty(); Checks whether collection contains elements or not. If no elements are present it returns false. boolean contains(Object o); Checks whether specified object is in collection or not. Return true if object is in collection. Iterator iterator(); Used to iterator over collection. No guarantee on order of elements iterated. boolean retainAll(Collection c); Removes all the elements which are not in specified collection. Returns only elements specified in collection removing other elements. Object[] toArray(); Returns an array of elements in collection.  **List the interfaces which extends collection interface ?**  1) List 2) Set 3) Queue 4) Deque ( From Java 6) 203) Explain List interface ? List interface extends collection interface used to store sequence of elements in collection. We can even store duplicate elements in list. We can insert or access elements in list by using index as we do in arrays. List is an ordered collection. The main difference between List and non list interface are methods based on position. 38 Some of the operations we can perform on List : 1) Adding an element at specified index. 2) Removing an element at specified index. 3) To get the index of element List contains some specific methods apart from Collection interface methods.  **Explain methods specific to List interface ?**  boolean addAll(int index, Collection c); This method inserts all the elements in specified collection to the list at specified position. E get(int index); This method returns an element at specified position in the list. E set(int index, E element); This method replaces the element at specified position in the list with the specified element. void add(int index, E element); This method inserts the specified element with the index specified. E remove(int index); This method removes the element at specified index and returns the element removed. int indexOf(Object o); indexOf() method returns the index of last occurrence of specified element. If there is no element in the list it removes the element. ListIterator listIterator(); Returns a list iterator of elements in list. List subList(int fromIndex, int toIndex); This method returns list of elements between indexes specified.  **List implementations of List Interface ?**  1) ArrayList 2) Vector 3) LinkedList  **Explain about ArrayList ?**  ArrayList is an ordered collection which extends AbstractList and implements List interface. We use ArrayList mainly when we need faster access and fast iteration of elements in list. We can insert nulls in to arraylist. Arraylist is nothing but a growable array. public class ArrayList extends AbstractList implements List, RandomAccess, Cloneable, java.io.Serializable{} From java 1.4 ArrayList implements RandomAccess interface which is a marker interface which supports fast and random access. Advantages : 1) Faster and easier access. 2) Used for Random access of elements. Drawbacks : 1) We cannot insert or delete elements from middle of list.  **Difference between Array and ArrayList ?**  Arrays are used to store primitives or objects of same type or variables that are subclasses of same type. ArrayList : It is an ordered collection which grows dynamically. In list we can insert nulls values and list allows duplicate elements. ARRAY ARRAY LIST 1) While creating array we have to know the size. 1) But it is not required to know size while creating ArrayList, because arraylist grows dynamically. 2) To put an element in to array we use the following syntax :String array[] = newString[5];array[1] = “java”;We must know specific location to insert an element in to 2) We can add element to arraylist with following syntax :List stringList = new ArrayList();stringList.add(“java”); 39 array. If we try to put element in index which is out of range we get ArrayIndexOutOfBounds Exception 3) Arrays are static 3) ArrayList is dynamic 4) We can store objects and primitives 4) We can store only primitives prior to 1.5 . From 1.5 we can store even objects also. 5) We have to manually write logic for inserting and removing elements. 5) Just a method call would add or remove elements from list. 6) Arrays are faster 6) Arraylist is slower. 7) Arraylist is implemented using arrays.  **What is vector?**  Vector is similar to arraylist used for random access. Vector is a dynamic array like arraylist. vector size increases or decreases when elements are added and removed . Vector is synchronized . vector and Hashtable are the only collections since 1.0. Rest of the collections are added from 2.0. public class Vectorextends AbstractListimplements List, RandomAccess, Cloneable, java.io.Serializable  **Difference between arraylist and vector ?**  Both ArrayList and vector grows dynamically. The differences between arraylist and vector are : 1) Arraylist is not synchronized and vector is synchronized. 2) Vector is legacy collection introduced in 1.0 and Arraylist introduced in java 2.0. Performance wise it is recommended to use arraylist rather than vector because by default vector is synchronized which reduces performance if only one thread accesses it.  **In which order the Iterator iterates over collection?**  The order in which Iterator will iterate the collection depends on the traversal order of collection. For example : for list traversal order will be sequential, and for set the order cannot be determined, and for sorted sorted set will sort the elements in sorted order. So it all depends on the collection in which order iterator iterates.  **Explain ListIterator and methods in ListIterator?**  List Iterator is similar to Iterator but ListIterator is bidirectional. We can traverse through the collection in either forward or backward direction. List Iterator extends Iterator and all the methods in Iterator will be there in ListIterator too with some additional methods . List Iterator doesn’t have current element .Position of List Iterator lies between two elements i.e previous element and next element. Features of ListIterator : 1) Traversal of List in either direction.  2) Modification of its elements.  3) Access to elements position.  Signature : public interface ListIterator extends Iterator { } ListIterator methods : Method Description Void add(E obj) Inserts element in to the list infront of the element returned by call to next() and after the element returned by call to next(). boolean hasNext(); Returns true if there are more elements in the list instead of throwing exception if there are no elements. E next(); Returns the next element . NoSuchElementException is thrown if there is no next element. boolean hasPrevious(); Returns true if there are elements when iterating list in reverse direction. E previous(); Returns the previous element in the list. int nextIndex(); Returns the index of the element returned by next() method. If there are no elements it returns the size of the list. int previousIndex(); Returns the index of the element returned by previous() method. If there are no elements it returns the size of the list. Returns -1 if the iterator is at beginning of list. void remove(); Removes the element that was returned by calling next() or previous(). An Illegal state Exception will be thrown if remove() is called before next() or previous(). void set(E e); This method replaces an element in the list with the specified element.  **Explain about Sets ?**  A set is a collection which does not allow duplicates. Set internally implements equals() method which doesn’t allow duplicates.Adding an duplicate element to a set would be ignored .Set interface is implemented in java.util.set package.Set interface does not have any additional methods . It has only collection methods. A set can contain atmost one null value. ArrayList is an ordered collection.In arraylists order remains same in which they are inserted. But coming to set it is an unordered collection. public interface Set extends Collection { } Important operations that can be performed on set : 1) Adding an element to set. 2) Removing an element from set. 3) Check if an element exist in set. 4) Iterating through set.  **Implementations of Set interface ?**  1) HashSet 2) Linked HashSet 3) TreeSet  **Explain HashSet and its features ?**  Hashset implements set interface and extends AbstractSet. Features of Hashset are : 1) It does not allow duplicates. 2) It does not gurantee ordering of elements. 3) It is unsorted and unordered set. 4) Performance wise it is recommended to use hashset when compared to other sets because it internally uses hashing mechanism. 5) Allows insertion of nulls. Note : For efficiency whenever objects are added to HashSet it need to implement the hashCode() method. public class HashSet extends AbstractSet implements Set, Cloneable, java.io.Serializable { }  **Explain Tree Set and its features?**  TreeSet implements navigableSet interface and extends Abstract set.It creates collection that uses tree for storage. Features of Treeset are : 1) It does not allow duplicates. 2) When we retrieve the elements in treeset we will get elements in sorted order. 42 public class TreeSet extends AbstractSet implements NavigableSet, Cloneable, java.io.Serializable { 217) When do we use HashSet over TreeSet? If we want to search for an element in collection and does not want any sorting order we go for HashSet. 82) When do we use TreeSet over HashSet? TreeSet is preferred 1) if elements are to be maintained in sorting order. 2) Fast insertion and retrieval of elements.  **What is Linked HashSet and its features?**  LinkedHashSet extends HashSet and implements Set interface. public class LinkedHashSet extends HashSet implements Set, Cloneable, java.io.Serializable { } Linked HashSet is similar to HashSet but in linked HashSet we maintain order but in HashSet we don’t maintain order. Maintaining order means elements will be retrieved in order which they are inserted.  **Explain about Map interface in java?**  A map is an association of key-value pairs. Both keys and values in map are objects. Features of map : 1) Maps cannot have duplicate keys but can have duplicate value objects.  **What is linked hashmap and its features?**  LinkedHashMap extends HashMap and implements Map.lLinked hashmap gurantees order of elements . Elements are retrieved in same order they are inserted.Linked HashMap uses internally double linked lists to keep insertion order. The differences between Hashmap and linked hashmap is 1) LinkedHashMap maintains the insertion order while HashMap doesnot maintain order. 2) HashMap if faster for insertion and deletion of elements when compared to linked hashmap. Linked hashmap is preferred only for faster iteration of elements. public class LinkedHashMap extends HashMap implements Map { }  **What is SortedMap interface?**  SortedMap extends Map interface.Sorted Map maintains sorted order of keys in a map. By default sorted map maintains natural ordering if we want custom order we can specify using comparator. public interface SortedMap extends Map { } 222) What is Hashtable and explain features of Hashtable? Hashtable was available before collection framework. When collection framework was started Hashtable extends Dictionary class and Map interface. Hashtable offers a convenient way of storing key/ value pairs. Hashtable does not allow nulls either keys or values. Hashtable is synchronized.  **Difference between HashMap and Hashtable?**  Difference HashMap Hashtable Synronization HashMap is not synchronized. Hashtable is synchronized. Nulls HashMap allows atmost one null key and any number of null values. Hashtable does not allow null values. Performance Since HashMap is not synchronized its performance is faster than Hashtable. Performance is slower when compared to HashMap. Introduction HashMap introduced starting from Hashtable is even before collection 43 collection framework. framework.  **Difference between arraylist and linkedlist**?  Difference Arraylist LinkedList Access Implements RandomAccess interface we can search randomly all the elements in the list. It extends Abstract sequential List interface which provides sequential access to elements. Searching and retrieval of elements Searching and retrieval of elements is fast since arraylist provides random access. Searching and retrieval of elements is slow because of sequential access to elements. Addition and removal of elements Adding and removal of elements in random positions is slow.For example if we want to add element to middle of the list we have to move the elements in the list and then we need to insert the element. Similarly for removing the element we need to follow the same thing. Adding and removal of elements in random positions is fast because there is no need of resizing the array just by updating the node structures with new addresses.  **Difference between Comparator and Comparable in java?** Sno Comparator Comparable 1. Defined in java.util package Defined in java.lang package. 2. Comparator interface is used when we want to compare two different instances Comparable is used to compare itself with other instance. 3. Comparator is used when we want custom sorting.Ex : If we take employee class sorting by employeeId is natural sorting. Comparable is used for natural sorting of objects.Ex : If we take employee class sorting by ename and age we can say as custom sorting. 4. Should override int compare(T o1, T o2) method which takes two instances. Should override public int compareTo(T o) method which takes one instance. 5. For sorting objects we use collections.sort(list,new Comparator); For sorting objects we use collections.sort(list);  **What is concurrent hashmap and its features ?**  Concurrent HashMap is implemented in java.util.concurrent package. Concurrent HashMap extends Abstract Map and implements concurrent Map. Concurrent HashMap is used in multi threaded environment. ]It is similar to Hashtable and synchronized version of hashmap but with minor differences. Concurrent HashMap does not allow null keys and values.  **Difference between Concurrent HashMap and Hashtable and collections.synchronizedHashMap?**  Locking Mechansim :ConcurrentHashMap uses completely different hashing mechanism called lock striping which offers better concurrency and scalability. The main advantage of this mechanism is better concurrency instead of synchronizing every method by using common lock which allows only one thread to access at a time, it allows better concurrency by allowing multiple threads to access. ConcurrentModificationException :ConcurrentHashMap provides iterators which doesnot throw concurrent modification exception which allows only one thread to access iterator, while synchronized map may throw concurrent modification exception. 228) Explain copyOnWriteArrayList and when do we use copyOnWriteArrayList? copyOnWriteArrayList is used in multithreaded environment. If we want to iterate over arraylist ,but the arraylist is updated by other threads to prevent concurrent modification exception we have two solutions : 1) First one is we need to synchronize the list manually by using collections.synchronized(list) and iterate over the list in synchronized block to avoid concurrent modification exception. 2) The second one is to use copyOnWriteArrayList which takes care of concurrency. The advantage of using copyOnWriteArrayList is no need to synchronize list explicitly. So when we use copyOnWriteArrayList when a thread modifies the list while the other thread was iterating it does not modify original list but creates a copy of list with modified contents so that the iterator won’t know the modifications made to original list.  **Explain about fail fast iterators in java?**  When iterator iterates over collection, collection should not be modified except by that iterator. Modification means collection cannot be modified by thread when other thread is iterating, if such modification happens a concurrent modification exception will be thrown.Such kind of iterators are fail fast iterators. Ex : ArrayList,HashSet,HashMap. Almost all the iterators implemented in collections framework are fail fast.  **Explain about fail safe iterators in java?**  Fail safe iterators are iterators which does not throw concurrent modification exception, when one thread modifies collection and other thread in the process of iterating the collection. It does not throw concurrent modification exception because when other thread was iterating it does not modify original list but creates a copy of list with modified contents so that the iterator won’t know the modifications made to original list. Ex : copyOnWriteArrayList Core java Serialization interview questions  **What is serialization in java?**  Serialization is the process of converting an object in to bytes, so that it can be transmitted over the network,or stored in a flat file and can be recreated later. Serialized object is an object represented as sequence of bytes that includes objects data, object type, and the types of data stored in the object.  **What is the main purpose of serialization in java?**  The main uses of serialization are : 1) Persistence: We can write data to a file or database and can be used later by deserializing it. 2) Communication : To pass an object over network by making remote procedure call. 3) Copying : We can create duplicates of original object by using byte array. 4) To distribute objects across different JVMs.  **What are alternatives to java serialization?**  XML based data transfer JSON based data transfer. XML based data transfer : We can use JIBX or JAXB where we can marshall our object’s data to xml and transfer data and then unmarshall and convert to object. JSON based transfer : We can use json to transfer data. 234) Explain about serializable interface in java? To implement serialization in java there is an interface defined in java.io package called serializable interface. Java.io.Serializable interface is an marker interface which doesnot contain any any methods. A class implements Serializable lets the JVM know that the instances of the class can be serialized. Syntax: public interface Serializable { }  **How to make object serializable in java?**  1) Our class must implement serializable interface.If our object contains other objects those class must also implement serializable interface. 2) We use ObjectOutputStream which extends OutputStream used to write objects to a stream. 3) We use ObjectInputStream which extends InputStream used to read objects from stream  **What is serial version UID and its importance in java?**  Serial version unique identifier is a 64 bit long value .This 64 bit long value is a hash code of the class name,super interfaces and member. Suid is a unique id no two classes will have same suid. Whenever an object is serialized suid value will also serialize with it. When an object is read using ObjectInputStream, the suid is also read. If the loaded class suid does not match with suid read from object stream, readObject throws an InvalidClassException.  **What happens if we don’t define serial version UID ?**  If we don’t define serial version UID JVM will create one suid for us. But it is recommended to have suid rather than JVM creating because at run time JVM has to compute the hashcode of all the properties of class. This process makes serialization low. We can’t serialize static fields one exception to this is suid where suid gets serialized along with the object. Ex :private static final long serialVersionUID = -5885568094444284875L;  **Can we serialize static variables in java?**  We can’t serialize static variables in java. The reason being static variable are class variables that belongs to a class not to object, but serialization mechanism saves only the object state not the class state.  **When we serialize an object does the serialization mechanism saves its references too?**  When we serialize an object even the object it refers must implement serializable then the reference objects also get serialized. If we don’t make reference objects serializable then we get NotSerializableException.  **If we don’t want some of the fields not to serialize How to do that?**  If we don’t want to serialize some fields during serialization we declare those variables as transient. During deserialization transient variables are initialized with default values for primitives and null for object references  **Object Oriented programming Interview Questions**  **1. What is Object Oriented Programming?**  Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.  **2. Why use OOPs?**   * OOPs allows clarity in programming thereby allowing simplicity in solving complex problems * Code can be reused through inheritance thereby reducing redundancy * Data and code are bound together by encapsulation * OOPs allows data hiding, therefore, private data is kept confidential * Problems can be divided into different parts making it simple to solve * The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms     **3. What are the main features of OOPs?**   * Inheritance * Encapsulation * Polymorphism * Abstraction   **4) What is Inheritance?**  **Answer:** Inheritance means one class can extend to another class. So that the codes can be reused from one class to another class. The existing class is known as the Super class whereas the derived class is known as a sub class.  **Example:**   |  | | --- | | Super class:  public class Manupulation(){  }  Sub class:  public class Addition extends Manipulation(){  } |   Inheritance is only applicable to the public and protected members only. Private members can’t be inherited.  **5) What is Encapsulation?**  **Answer: Purpose of Encapsulation:**   * Protects the code from others. * Code maintainability.   **Example:**  We are declaring ‘a’ as an integer variable and it should not be negative.   |  | | --- | | public class Addition(){  int a=5;  } |   If someone changes the exact variable as “***a = -5”***then it is bad  **6) What is Polymorphism?**  **Answer:** Polymorphism means many forms.  A single object can refer to the super-class or sub-class depending on the reference type which is called polymorphism.  **Example:**   |  | | --- | | Public class Manipulation(){ //Super class  public void add(){  }  }  public class Addition extends Manipulation(){ // Sub class  public void add(){  }  public static void main(String args[]){  Manipulation addition = new Addition();//Manipulation is reference type and Addition is reference type  addition.add();  }  } |   Using the Manipulation reference type we can call the Addition class “add()” method. This ability is known as Polymorphism. Polymorphism is applicable for **overriding**and not for **overloading**.  **7) What is meant by Method Overriding?**  **Answer: Method overriding happens if the sub-class method satisfies the below conditions with the Super-class method:**   * Method name should be the same * The argument should be the same * Return type should also be the same   The key benefit of overriding is that the Sub-class can provide some specific information about that sub-class type than the super-class.  **Example:**   |  | | --- | | public class Manipulation{ //Super class  public void add(){  ………………  }  }    Public class Addition extends Manipulation(){  Public void add(){  ………..  }  Public static void main(String args[]){  Manipulation addition = new Addition(); //Polimorphism is applied  addition.add(); // It calls the Sub class add() method  }  } |   **addition.add()**method calls the add() method in the Sub-class and not the parent class. So it overrides the Super-class method and is known as Method Overriding.  **8) What is meant by Overloading?**  **Answer:** Method overloading happens for different classes or within the same class.  **For method overloading, sub-class method should satisfy the below conditions with the Super-class method (or) methods in the same class itself:**   * Same method name * Different argument types * There may be different return types   **Example:**   |  | | --- | | public class Manipulation{ //Super class  public void add(String name){ //String parameter  ………………  }  }    Public class Addition extends Manipulation(){  Public void add(){//No Parameter  ………..  }  Public void add(int a){ //integer parameter    }  Public static void main(String args[]){  Addition addition = new Addition();  addition.add();  }  } |   Here the add() method has different parameters in the Addition class is overloaded in the same class as with the super-class.  **Note:** Polymorphism is not applicable for method overloading.  **9) What is meant by Interface?**  **Answer:** Multiple inheritances cannot be achieved in java. To overcome this problem the Interface concept is introduced.  An interface is a template which has only method declarations and not the method implementation.  **Example:**   |  | | --- | | Public abstract interface IManupulation{ //Interface declaration  Public abstract void add();//method declaration  public abstract void subtract();  } |  * All the methods in the interface are internally **public abstract void**. * All the variables in the interface are internally **public static final** that is constants. * Classes can implement the interface and not extends. * The class which implements the interface should provide an implementation for all the methods declared in the interface.  |  | | --- | | public class Manupulation implements IManupulation{ //Manupulation class uses the interface  Public void add(){  ……………  }  Public void subtract(){  …………….  }  } |   **10) What is meant by Abstract class?**  **Answer:** We can create the Abstract class by using the “Abstract” keyword before the class name. An abstract class can have both “Abstract” methods and “Non-abstract” methods that are a concrete class.  **Abstract method:**  The method which has only the declaration and not the implementation is called the abstract method and it has the keyword called “abstract”. Declarations ends with a semicolon.  **Example:**   |  | | --- | | public abstract class Manupulation{  public abstract void add();//Abstract method declaration  Public void subtract(){  }  } |  * An abstract class may have a non- abstract method also. * The concrete Subclass which extends the Abstract class should provide the implementation for abstract methods.   11**) Difference between overriding and overloading in java?**  Overriding Overloading In overriding method names must be same In overloading method names must be same Argument List must be same Argument list must be different atleast order of arguments. Return type can be same or we can return covariant type. From 1.5 covariant types are allowed Return type can be different in overloading. We cant increase the level of checked exceptions. No restrictions for unchecked exceptions In overloading different exceptions can be thrown. A method can only be overridden in subclass A method can be overloaded in same class or subclass Private,static and final variables cannot be overridden. Private , static and final variables can be overloaded. In overriding which method is called is decided at runtime based on the type of object referenced at run time In overloading which method to call is decided at compile time based on reference type. Overriding is also known as Runtime polymorphism, dynamic polymorphism or late binding Overloading is also known as Compile time polymorphism, static polymorphism or early binding.  12) **What is ‘IS-A ‘ relationship in java?**  ‘is a’ relationship is also known as inheritance. We can implement ‘is a’ relationship or inheritance in java using extends keyword. The advantage or inheritance or is a relationship is reusability of code instead of duplicating the code. Ex : Motor cycle is a vehicle Car is a vehicle Both car and motorcycle extends vehicle.  13) **What is ‘HAS A’’ relationship in java?**  ‘Has a ‘ relationship is also known as “composition or Aggregation”. As in inheritance we have ‘extends’ keyword we don’t have any keyword to implement ‘Has a’ relationship in java. The main advantage of ‘Has-A‘ relationship in java code reusability.  14) **Difference between ‘IS-A’ and ‘HAS-A’ relationship in java?**  IS-A relationship HAS- A RELATIONSHIP Is a relationship also known as inheritance Has a relationship also known as composition or aggregation. For IS-A relationship we uses extends keyword For Has a relationship we use new keyword Ex : Car is a vehicle. Ex : Car has an engine. We cannot say Car is an engine The main advantage of inheritance is reusability of code The main advantage of has a relationship is reusability of code.  15 ) **What are packages in java?**  Package is a mechanism to group related classes ,interfaces and enums in to a single module. Package can be declared using the following statement : Syntax : package Coding Convention : package name should be declared in small letters. package statement defines the namespace. The main use of package is 1) To resolve naming conflicts 2) For visibility control : We can define classes and interfaces that are not accessible outside the class.  **16) Can we have more than one package statement in source file ?**  We can’t have more than one package statement in source file. In any java program there can be atmost only 1 package statement. We will get compilation error if we have more than one package statement in source file. 41) Can we define package statement after import statement in java? We can’t define package statement after import statement in java. package statement must be the first statement in source file. We can have comments before the package statement.  17) **What are identifiers in java?**  Identifiers are names in java program. Identifiers can be class name, method name or variable name. Rules for defining identifiers in java: 1) Identifiers must start with letter,Underscore or dollar($) sign. 2) Identifiers can’t start with numbers . 3) There is no limit on number of characters in identifier but not recommended to have more than 15 characters 4) Java identifiers are case sensitive. 5) First letter can be alphabet ,or underscore and dollar sign. From second letter we can have numbers . 6) We should’nt use reserve words for identifiers in java.  18) **What are access modifiers in java?**  The important feature of encapsulation is access control. By preventing access control we can misuse of class, methods and members. 16 A class, method or variable can be accessed is determined by the access modifier. There are three types of access modifiers in java. public,private,protected. If no access modifier is specified then it has a default access.  19) **What is the difference between access specifiers and access modifiers in java?**  In java we have access specifiers as public,private,protected and default and access modifiers as static, final. But there is no such divison of access specifiers and access modifiers in java. In Java we have access modifiers and non access modifiers. Access Modifiers : public, private, protected, default Non Access Modifiers : abstract, final,.  20) **What access modifiers can be used for class ?**  We can use only two access modifiers for class public and default. public: A class with public modifier can be visible 1) In the same class 2) In the same package subclass 3) In the same package nonsubclass 4) In the different package subclass 5) In the different package non subclass. default : A class with default modifier can be accesed 1) In the same class 2) In the same package subclass 3) In the same package nonsubclass 4) In the different package subclass 5) In the different package non subclass.  21) **Explain what access modifiers can be used for methods?**  We can use all access modifiers public, private,protected and default for methods. public : When a method is declared as public it can be accessed 6) In the same class 7) In the same package subclass 8) In the same package nonsubclass 9) In the different package subclass 10) In the different package non subclass. default : When a method is declared as default, we can access that method in 1) In the same class 2) In the same package subclass 3) In the same package non subclass We cannot access default access method in 1) Different package subclass 2) Different package non subclass. protected : When a method is declared as protected it can be accessed 1) With in the same class 2) With in the same package subclass 3) With in the same package non subclass 4) With in different package subclass It cannot be accessed non subclass in different package. private : When a method is declared as private it can be accessed only in that class. It cannot be accessed in 1) Same package subclass 2) Same package non subclass 3) Different package subclass 4) Different package non subclass.  22) **Explain what access modifiers can be used for variables?**  We can use all access modifiers public, private,protected and default for variables. public : When a variables is declared as public it can be accessed 1) In the same class 2) In the same package subclass 3) In the same package nonsubclass 4) In the different package subclass 5) In the different package non subclass. default : When a variables is declared as default, we can access that method in 1) In the same class 2) In the same package subclass 3) In the same package non subclass We cannot access default access variables in 4) Different package subclass 5) Different package non subclass. protected : When a variables is declared as protected it can be accessed 1) With in the same class 2) With in the same package subclass 3) With in the same package non subclass 4) With in different package subclass It cannot be accessed non subclass in different package. private : When a variables is declared as private it can be accessed only in that class. It cannot be accessed in 1) Same package subclass 2) Same package non subclass 3) Different package subclass 4) Different package non subclass.  23) **What is final access modifier in java?**  final access modifier can be used for class, method and variables. The main advantage of final access modifier is security no one can modify our classes, variables and methods. The main disadvantage of final access modifier is we cannot implement oops concepts in java. Ex : Inheritance, polymorphism. final class : A final class cannot be extended or subclassed. We ar e preventing inheritance by marking a class as final. But we can still access the methods of this class by composition. Ex: String class final methods: Method overriding is one of the important features in java. But there are situations where we may not want to use this feature. Then we declared method as final which will print overriding. To allow a method from being overridden we use final access modifier for methods. final variables : If a variable is declared as final ,it behaves like a constant . We cannot modify the value of final variable. Any attempt to modify the final variable results in compilation error. The error is as follows “final variable cannot be assigned.”  24) **Explain about abstract classes in java?**  Sometimes we may come across a situation where we cannot provide implementation to all the methods in a class. We want to leave the implementation to a class that extends it. In such case we declare a class as abstract.To make a class abstract we use key word abstract. Any class that contains one or more abstract methods is declared as abstract. If we don’t declare class as abstract which contains abstract methods we get compile time error. We get the following error. “The type must be an abstract class to define abstract methods.” Signature ; abstract class { } For example if we take a vehicle class we cannot provide implementation to it because there may be two wheelers , four wheelers etc. At that moment we make vehicle class abstract. All the common features of vehicles are declared as abstract methods in vehicle class. Any class which extends vehicle will provide its method implementation. It’s the responsibility of subclass to provide implementation. The important features of abstract classes are : 1) Abstract classes cannot be instantiated. 2) An abstract classes contains abstract methods, concrete methods or both. 3) Any class which extends abstract class must override all methods of abstract class. 4) An abstract class can contain either 0 or more abstract methods.  **Spring Boot :**  **Annotations In Spring Boot :**  **@Required:** It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitilizationException**.  **@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.  **@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.  **@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.  **@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.  **@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.  **@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.  **@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.  **@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.  **@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication**.  **@SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.  **@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. We use it with the class as well as the method.  **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**  **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**  **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**  **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**  **@PatchMapping:** It maps the **HTTP PATCH**requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**  **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.  **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.  **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.  **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.  **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.**For each detail in the header, we should specify separate annotations. We can use it multiple time in a method  **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody**annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.  **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.   1. What is Spring? Why to use it?   • The Spring Framework (Spring) is an open-source application framework that provides infrastructure support for developing Java applications. One of the most popular Java Enterprise Edition (Java EE) frameworks, Spring helps developers create high performing applications using plain old Java objects (POJOs). • Without a Spring Framework, application code tends to be tightly coupled (interdependent), which is not considered good coding practice. Loose coupling is ideal because loosely coupled components are independent, meaning changes in one will not affect the operation of others.   1. What are the features of Spring framework?   • It is light weight (JARs are smaller in size) & open source. • It is complete end (applicable for all layers in application) & modular framework (applicable for particular layer in application). • We can achieve loose coupling. (Provides features of IoC and DI) • Non-invasive framework (doesn’t force to extend or implement any base class or interface). • We can develop easy to test kind of applications. • Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc.   1. Does Spring contain in-built server?   • Spring don’t have in-built server. We have to add it explicitly.   1. What is tight coupling and loose coupling?   • Tight Coupling ---> Change in one class forces to change in other classes • Loose Coupling ---> Change in one class doesn’t forces to change in other classes  5. Enlist modules in Spring framework? • Core Container ---> This module provides IoC and DI feature. It also contains BeanFactory, ApplicationContext. • Data Access/Integration ---> Provides support to interact with DB and to integrate with ORM (Object Relationship Mapping) like Hibernate. • Web ---> Provides basic support for web development. It has WebApplicationContext built on ApplicationContext of a core container which provides complete MVC (Model View Controller) implementation to develop presentation layer.  6. What is IoC? • The IoC container creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time. • It means object creation is not a task of programmer it will be handled by IoC container. Called as Inversion of Control because represents the inversion responsibility of object creation, initialization and destruction (object life cycle) from application to Spring Container. • Normal Object creation----> using new keyword In Spring, Core Container provides the required dependency object. • It converts object from tight coupling to loose coupling which is achieved by dependency injection.  7. What is purpose of Spring container? • It takes care of object creation, initialization and managing object dependencies.  8. What is Dependency? Types of Dependencies. • Data member or properties are dependencies. • Types--> Primitive, Object/secondary, Collection  9. What is Dependency Injection? Types of Dependency Injection. • Assigning values to those data members(dependencies) is Dependency Injection. • Can be achieved in two ways--->1. Setter Based 2.Constructor Based  10.Difference between setter based and constructor-based DI? Setter based Constructor-based 1.Allows partial injection 1. Doesn’t allow partial injection 2.Can override constructor-based DI 2.Cannot override setter-based DI 3.Mutable 3.Immutable  11.What is container? Different types of container. • Container is the one which is responsible for object life cycle in Spring. • Types----> 1. Core container------>BeanFactory (interface) 2.JEE container ------>ApplicationContext (Child interface of BeanFactory)  12.Difference between BeanFactory and ApplicationContext container? BeanFactory ApplicationContext 1.Implemented by XMLBeanFactory 1.Implemented by WebApplicationContext 2.Doesn’t support annotation concept 2.Supports annotation concept 3.Doesn’t support internationalization 3.Supports internationalization 4.Lazy loading of beans 4.Eager/aggressive loading of beans 5.Supports desktop-based application 5.Supports both desktop-based and webbased applications Note ---> Each bean has a unique id.  13.What is Lazy and Eager loading? • Lazy Loading --->instantiates bean only when called. • Eager Loading --->instantiates bean at the time of container start.  14.What is Scope in Spring? What are its types? • When the Spring Framework creates a bean, it associates a scope with the bean. A scope defines the runtime context within which the bean instance is available. • By default, Spring bean scope is Singleton. • Types---> 1.Singleton ---> - Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean. - BeanFactory: Lazy ApplicationContext: Eager 2.Prototype ---> - A new instance will be created for a single bean definition every time a request is made for that bean. - BeanFactory: Lazy ApplicationContext: Lazy 3.Request (Valid for Spring based applications, used for http request) 4.Session (Valid for Spring based applications, used for http session) 5.Application 6.Websocket  15.How to make ApplicationContext as Lazy loading? • By declaring ApplicationContext bean scope as Prototype.  16.What is an AutoWiring? How to enable AutoWiring? • Autowiring feature of spring framework enables you to inject the secondary dependencies implicitly (automatically). It internally uses setter or constructor injection. • Autowiring can't be used to inject primitive and string values. It works with reference only. • Can be achieved by --->  17.By default, AutoWire supports which autowire type? • By default, it uses byType.  18.How to enable autowire in .xml file? • By default, it is disabled. • Add in bean configuration xml file  19.What is front-controller or Dispatcher servlet? Its uses? • It manages entire process. Operation starts from Dispatcher-servlet so, no need of main method. • It finds appropriate class as per request 20.What is Java based? • In this we don’t use .xml, instead of that we use java class. byName - Dependency is injected according to name of the bean. - In this scenario Has-A variable name in POJO and bean name must be same. byName byType - This option enables the autowire based on bean type. - In this scenario Has-A variable name in POJO and bean name can be different but there must be only one bean of a type. byType constructor - Injects dependency by calling constructor. Internally uses byType. Bean Type Bean Name  21.What is scope combination?  22.what is lookup method? When to use?  23.What is Stereotype Annotation? Explain them? • @Controller: Identifies class as a controller class & marks it as a bean. • @Service: Identifies class as a service class & tells it has business logic. • @Repository: Identifies class as a DAO layer class & tells it has a database connection.  24.Can we interchange any of above stereotype annotations? • Yes, we can shuffle @Repository & @Service. • It is just to understand other programmers to identify business logic class and DAO layer class. • But we can’t replace @Controller class as it marks as bean.  25.What are Spring validations? • The Spring MVC Validation is used to restrict the input provided by the user. It can validate both server-side as well as client-side applications.  26.What is Spring MVC? • Helps to develop web-based applications. Helps to separate different aspects of programming. e.g., input logic, business logic, UI logic Model= binds the called data of an application View= generates the html output Controller=It processes the user request, and built model passes it to view.  27.Execution flow of MVC? DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views. o Model - A model contains the data of the application. A data can be a single object or a collection of objects. o Controller - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller. o Front Controller - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application. o View - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker. Spring provides view resolvers, which enable you to render (to show) models in a browser without tying you to a specific view technology. As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller. o The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller. o The controller returns an object of ModelAndView. o The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.  28.Difference between URL and URI? • A URI is a sequence of characters that identifies a web resource by location, name, or both available on the internet. Whereas, URL is a sequence of characters that only identifies the location of a resource available on the internet.  URI URL 1. URI is an acronym for Uniform Resource Identifier. URL is an acronym for Uniform Resource Locator. 2. URI contains two subsets, URN, which tell the name, and URL, which tells the location. URL is the subset of URI, which tells the only location of the resource. 3. All URIs cannot be URLs, as they can tell either name or location. All URLs are URIs, as every URL can only contain the location. 4. A URI aims to identify a resource and differentiate it from other resources by using the name of the resource or location of the resource. A URL aims to find the location or address of a resource on the web. Endpoint vs URL The term endpoint is focused on the URL that is used to make a request. The term resource is focused on the data set that is returned by a request. Now, the same resource can often be accessed by multiple different endpoints. Also, the same endpoint can return different resources, depending on a query string. Each content present in the database is considered as a resource(objects). A resource can be text file, an image or any dynamic data. URI is used to identify each resource of REST (part of a Webservices) architecture. URI-Format Example ://// <https://en.wikipedia.org/wiki/ReST-api>  **Spring Boot Questions: -**  29.What is SpringBoot? Why do we mean by Stand-alone application? • Spring Boot is the extension of the Spring framework and provides a faster way to build applications. Spring Boot is an open-source Java web framework used for Rapid Application Development (to build stand-alone application with minimum configurations). The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase. • A standalone application is an application that runs locally on the device and doesn't require anything else to be functional. All the logic is built into the app, so it doesn't need an internet connection nor any other services installed.  30.Features of SpringBoot. 1. Provides embedded tomcat, jetty server. 2. Provides auto-configuration feature. 3. Provides production ready features (Actuators concept) such as matrix, healthcheck, info etc. 4.No requirement for XML configuration. Provides POM.xml starter file. 5.Provides Profiles feature.  31.Why to use SpringBoot over Spring? • Spring Boot not only improves productivity but also provides a lot of conveniences to write your own business logic. 1. Easy to understand and develop spring applications. 2. Spring Boot is nothing but an existing framework with the addition of an embedded HTTP server and annotation configuration which makes it easier to understand and faster the process of development. 3. Increases productivity and reduces development time. 4. Minimum configuration. We don’t need to write any XML configuration, only a few annotations are required to do the configuration. Below are some key points- • Starter POM. • Auto Configuration. • Component Scanning. • Embedded server and Actuators.  32.How to develop SpringBoot application? 1.Maven 2.STS (Spring Tool Suite) 3.Spring Initializer  33.What are main components Spring Boot Framework? • Spring Boot Framework has mainly four major Components 1.Spring Boot Starters. 2.Spring Boot Autoconfiguration. 3.Spring Boot CLI. 4.Spring Boot Actuator. 34.What is Spring Initializer? • Spring Initializer is a web application that helps you to create an internal project structure for you. So, you do not have to manually set up the structure of the project, instead, you can use this feature.  35.Explain the steps to create a Spring Boot project using Spring Initializer? • Spring Initializer is a web tool provided by Spring. With the help of this tool, you can create Spring Boot projects by just providing project details. The following steps need to be followed to create a Spring Boot project using Spring Initializer: 1. Choose the maven project and the required dependencies. Then, fill in the other required details like Group, Artifact, and then click on Generate Project. 2. Once the project is downloaded, extract the project onto your system 3. Next, you have to import this project using the import option on the Spring Tool Suite IDE 4. While importing the project, remember that you have to choose the project type to be Maven and the source project should contain the pom.xml file. Once, all the above steps are followed you will see that the Spring Boot project is created with all the required dependencies.  36.Explain the steps to connect Spring Boot project with JDBC(MySQL)? Step 1: Create a database in MySQL Step 2: Add the JDBC, MySQL and web dependencies in the application. Step 3: Need to configure the database into application.properties.  37.What is SpringBoot CLI and its use? • Spring Boot CLI (Command Line Interface) is a Spring Boot software to run and test Spring Boot applications from command prompt. When we run Spring Boot applications using CLI, then it internally uses Spring Boot Starter and Spring Boot Autoconfiguration components to resolve all dependencies and execute the application.  38.Difference between Spring and SpringBoot. Spring SpringBoot 1.No auto-configuration feature. 1. Provides auto-configuration. 2.No in-built server. 2. In-built tomcat server. 3.Requires xml configuration 3. No need of xml configuration. 4.More boiler plate code. 4. Reduced boiler plate code.  39.Internal Work flow of SpringBoot application. • The entry point is a class which contains @SpringBootAnnotation and has main method. • SpringBoot scan all the components included in the project using annotation @ComponenetScan and generates Beans itself. • Using @EnableAutoConfiguration annotation SpringBoot configures the application automatically.  41.What is SpringBoot starter? Why it is useful? • SpringBoot starter comprises of number of templates which provides a Rapid Application Development (RAD). It contains combination of relative transient dependencies needed to start particular functionality. • Starter is a jar file which predominantly solves the auto-dependency resolution in a SpringBoot application. • Syntax: spring-boot-starter-X, here is X is required dependency. E.g.- spring-boot-starter-web, so here SpringBoot internally downloads and add all web(X) associated dependencies to the application.  42.Purpose of SpringApplication.run()? • SpringApplication.run() bootstraps a spring application as a stand-alone application from the main method. It creates an appropriate ApplicationContext instance and load beans. It also runs embedded Tomcat server in Spring web application.  43.Does tag add any dependencies? • No, it doesn’t add any dependency. It helps to maintain the application. • ----> One reason to use a parent is that you have a central place to store information about versions of artifacts, compilersettings etc. • ---->A dependency is libraries you need to get your code to compile. This can be your own code, or libraries such as Apache Commons. A parent contains information, but nothing to actually build, that is shared between a number of your projects.  44.What is devtools? • DevTools stands for Developer Tool. The aim of the module is to improve the development time while working with the Spring Boot application. Spring Boot DevTools pick up the changes and restart the application automatically (no need to re-run).  45.What is SpringBoot dependency management? • Spring Boot dependency management is used to manage dependencies and configuration automatically without specifying the version for any of those dependencies.  46.What is HotSwapping? • Hot swapping means you can modify the source code and see the changes immediately, simply by reloading the page in the web browser, without rebuilding your project and have to restart the web server.  47.Which is default port in SpringBoot? • Default port -----> 8080  48.Can we change port number of embedded Tomcat server? How? • Custom port ----->server.port=9999 • If server.port=0 then it will automatically assign any available port.  49.Can we run SpringBoot Application on custom port? (YES)  50.Can we replace embedded Tomcat server? • Yes. By using starter dependency in POM.xml. Exclude tomcat and add required dependency.  51.Can we disable default web server in SpringBoot? • Yes. By using application.properties to configure the web application type. spring.main.web-application-type=none  52.What is an actuator? Its uses, advantages. • An actuator is a feature of SpringBoot that helps to monitor and manage the application when pushed to production environment. These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering, and many more that are automatically applied to our application. • It’s a tool which has http endpoints.  53.How to activate actuator? What do we mean by http endpoints? • To enable the spring actuator feature, we need to add the dependency of “spring-boot-starter-actuator” in pom.xml • Spring Boot Actuator Endpoints lets us monitor and interact with our application.  54.How to run actuator endpoints and server on different ports? • For Actuator ----->management.server.port=9999 Management.endpoints.health.show-details=true On chrome hit----> http://localhost: 9999/actuator/ health • For Server-----> server.port=8800  55.How can we access actuator properties using custom URL? • Default endpoint -----> http://localhost: 9999/actuator/ add “management.endpoints.web.base-path=/customEndpointName” to application.properties • Hit Custom endpoint-----> http://localhost: 9999/customEndpointName/  57.What is profiles concept? Its uses, advantages. • While developing the application we deal with multiple environments such as dev, maintenance, Prod, and each environment requires a different configuration. • Even if DBMS is the same across all the environments, the URLs will be different. To make this easy and clean, Spring has the provision of Profiles to keep the separate configuration of each environment and those segregated configurations are available to that particular environment only. (Properties of dev available to dev only)  58.How to disable AutoConfiguration feature for a particular class? • Using exclude attribute @EnableAutoConfiguration(exclude = {XYZ.class}) @SpringBootApplication(exclude = {XYZ.class})  59.Difference between @RestController and @Controller. • @Controller map the model object to view or template and make it human readable but @RestController simply returns the object and object data is directly written in HTTP response as JSON or XML.  61.Difference between @RequestMapping and @GetMapping. • @RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation. Whereas @GetMapping is only an extension of @RequestMapping which helps you to improve on clarity on request. @Component @Endpoint(id="customActuator") public class CustomEndPoint { @ReadOperation public String test() { return "test"; } } http://localhost: 9999/actuator/customActuator Output: test @ReadOperation=Get method @Writeoperation=Post method  62.How to use property defined in an application.properties file into java class? • Use @Value (springframework.beans) to access value associated with application.properties E.g., @Value(“${server.port }”) public String serverPort;  63.How to get list of all Beans? • Spring Boot actuator “/beans” is used to get the list of all the spring beans in your application. e.g., http://localhost: 8200/actuator/ beans 64.About .yaml file and .properties file.  65.What is Spring Data JPA? (Java Persistence API) • Provides functionality for connection between application and relational database. It has 3 repository interfaces in that • CrudRepository (It is parent interface, used for CRUD operations) • PagingAndSortingRepository (It extends CrudRepository, used for Paging & Sorting purpose) • JPARepository (It extends PagingAndSortingRepository, used for all above and its additional services)  66.How to create custom query? • Using @Query annotation. E.g., @Query("select c from City c where c.name like %?1") List findByNameEndsWith(String chars);  67.How to Configure Hibernate in SpringBoot? application.properties spring.datasource.driver-class-name=com.mysql.jdbc.Driver spring.datasource.url=jdbc:mysql://localhost:3306/sbcurd spring.datasource.username=root spring.datasource.password=root spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL55Dialect spring.jpa.show-sql=true spring.jpa.hibernate.ddl-auto=update  69.Difference between @RequestBody and @ ResponseBody 70.What is the difference between @component and @bean? • @Component is a class level annotation whereas @Bean is a method level annotation and name of the method serves as the bean name. • @Component need not to be used with the @Configuration annotation where as @Bean annotation has to be used within the class which is annotated with @Configuration  72.Disadvantages of SpringBoot? • When we provide starters into POM.xml it downloads lots of dependencies from central repository to .m2 folder, at some point many of those dependencies may not be useful. So increases the size of an application. • Suitable for micro-services but not for large or monolithic web services.  73.What is AutoConfiguration? How it helps? Why SpringBoot is called Opinionated? • AutoConfiguration – automatically configures Spring application based on the dependencies added in POM.xml. • Spring Boot is opinionated because it follows the opinionated(ideal) default configuration that reduces developer efforts to configure the application. • Opinionated – a way of approaching a business process is inherently better and provides a software crafted around that approach.  **Webservices: -**  1. What is Webservices? • It is a software system that is specially designed to propagate communication between the client and server applications on WWW (World Wide Web). • It is the method, medium or way of communication among two or more devices over a network. It allows multiple applications built on different programming languages to communicate with each other without any trouble. It uses the internet for direct application-to-application interaction, and also allows you to expose business logic using API. • It is a software system for the interoperable machine to machine communication. • It is a collection of standards or protocols for exchanging information between two devices or application.  2. Advantages of Webservices? • Interoperability: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language and on any OS. Web service can be written in java programming and client can be PHP and vice versa. • Reusability: One web service can be used by many client applications at the same time. • Loose Coupling: Web services client code is totally independent with server code, so we have achieved loose coupling in our application. • Easy to deploy and integrate. • Multiple service versions can be running at same time. • Supports the transparent exchange of data to facilitate business integration. • Supports Remote Procedure Call (RPC), a powerful technique for constructing distributed, client-server-based applications.  3. What are the types of webservices? SOAP (Simple Object Access Protocol) Web Services: It is also referred to as transport-independent messaging protocol whose main purpose is to transfer a message, and is based on XML protocol. RESTful (Representational State Transfer) Web Services: It is developed to fulfil the shortcomings of SOAP and to make the web services more effective. SOAP ReST  1. It supports XML data format only. It supports XML, HTML, Plain text, JSON, etc. data formats  2. It is protocol based It is architecture based.  3. It has its own security. Need to add security.  4. It is less preferred. It is more preferred.  5. Need JAX-WS API to implement SOAP. Need JAX-RS API to implement REST. -JSP---> Java Servlet Pages -json---> JavaScript Object Notation (data gets stored in key value pair)  4. How to create Restful webservices? • Add REST dependency in pom.xml. (spring-boot-starter-data-rest) • Add @RestController annotation for controller class. • Map the methods written in controller class. Each content present in the database is considered as a resource(objects) in ReST. A resource can be text file, an image or any dynamic data. URI is used to identify each resource of REST architecture. URI-Format Example ://// <https://en.wikipedia.org/wiki/ReST-api>  5. What is REST? Why to use it? • REST is the acronym for Representational State Transfer. REST is an architectural style for developing applications that can be accessed over the network. • REST is a stateless client-server architecture where web services are resources and can be identified by their URIs (Uniform Resource Identifiers). It uses http protocol to fetch those resources. • We choose Restful web service because - Simple and easy to implement and test. - It is programming language and platform (OS) independent - Support different formats such as JSON, XML, HTML, etc - Faster and provide better performance. - It provides caching mechanism to minimize server calls for the same type of repeated requests. The protocol used by REST web services to interact with server is HTTP. Some of the HTTP methods----> GET: Used to get and read a resource (retrieve data). POST: Used to create a new resource (insert data). PUT: Used to update existing resources (update data). DELETE: Used to delete the resource. PATCH: Used to apply partial modifications to a resource. • Medium of communication between client and server is called “messaging”.  6. What is statelessness w.r.t. ReST? Advantages. • Statelessness is basically a condition or restriction where RESTful web services are not allowed to keep a client state on the server as per the REST architecture. Clients are responsible to pass their context to the server. To process the client’s request, the server then further stores this context. E.g., if you get data using Get method, but if you want to fetch same data again then we need to use Get method again to get the response and every time it will be considered as a new request. • Advantages: -No need to maintain previous interactions with clients. -Independent treatment of each method request. -Less complexity and simplified application design. • Example: Simple GET Request using NodeJS  7. What is the use of ARC and Postman tools? • To test the webservice • Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.  8. Explain http status codes. REST API generally returns the following status codes in HTTP response: • 200 OK • 201 Created • 202 Accepted • 302 Found • 400 Bad Request • 401 Unauthorized • 404 Not Found • 405 Method Not Allowed • 409 Conflict • 500 Internal Server Error  9. Difference between webservices and API webservices API 1. It must use a network. It can be online or offline. 2. All web services are APIs. All APIs are not web services. 3. It is interaction between applications over the network. It is an interface between two different applications. 4. Web service uses styles like REST, SOAP for communication. API can be used for any style of communication. 5. Web Services does not have a light-weight architecture. API has a light-weight architecture.  10.Difference between @PathParam and @PathVariable @PathVariable @PathParam 1. It is from spring framework. It is from JAX-RS. 2. It will work in spring MVC and REST It will work in REST only. 3. It is annotation on a method argument to bind it to the value of a URI template variable. It is a parameter annotation which allows you to map variable URI path fragments into your method call.  11.What is RestTemplate? What is its use? • Rest Template is used to create applications that consume RESTful Web Services. (Provides communication between two templates). Basically, used to communicate, access and operate produce data using consume.(Refer program)  12.Drawbacks of WebService? How to overcome from it? • Lack of state: most web applications require stateful mechanisms. Suppose you purchase a website which has a mechanism to have a shopping cart. It is required to know the number of items in the shopping cart before the actual purchase is made. This burden of maintaining the state lies on the client, which makes the client application heavy and difficult to maintain. • Last of security: REST doesn’t impose security such as SOAP. That is the reason REST is appropriate for public URLs, but it is not good for confidential data passage between client and server. • Monolithic Architecture  14.Components of http? • An HTTP request is divided into three parts: Request line, header and body.  15.Name the web service method that is read-only, and is idempotent? • The web service method that is read-only and safe is the GET method, and the web service methods that are idempotent are PUT and DELETE operations. Post is not idempotent. • Idempotent refers to the operations whose results will always be the same even if these operations are invoked so many times. That is if you hit any server with same request for any number of times then the every time the response should be same (no change in memory for that particular resource). e.g., With each Post operation there is change in the database memory allocation so it is not an idempotent but if we take Put operation there will not be any change in the databases memory even if you use it any number of times so, this is idempotent. Delete is also idempotent as if you delete a particular resource, you cannot delete the same resource again so, there will not be change using same method again.  16.What is Consumed or Produced Restful web service? Explain? • E.g., makemytrip(consume) accesses all airlines(produce) data to show availability of tickets for various location-locations. So here consume access produce data to show availability of services to the user and provides interfaces to book the tickets and after booking the ticket it updates both consume and produce data for that booking resources. That’s why user get mail from both produce(particular airline) as well as consume(makemytrip). Remember here consume isn’t scheduling the flights it is just accessing the produce and after the use of service it is updating the data for both(that is produce must know user details for that booking ) • Above is just an example. But consume can access produce as per need Note: Produce and Consume can have different databases  18.What is broken API? If we hit an API and if that API is not mapped in the controller, then such APIs are called broken API. (Status code ---> 404 not found) 19.How to convert java object to json and vice versa? • Object to json----> 1. Add gson dependency in POM.xml 2. new Gson().toJson(object) • Json to object----> 1. Add gson dependency in POM.xml 2. Gson g= new Gson(); Student s = g.fromJson(jsonString, Student.class);  USE OF…..  @annotations:  1. @SpringBootApplication Marks the class as a Spring Boot annotated class which contains main (). [@EnableAutoConfiguration + @ComponentScan + @Configuration] @EnableAutoConfiguration: Triggers autoconfiguration when dependency added in POM.xml. @ComponentScan: Scans packages and sub packages for Spring Beans. @ComponentScan (basePackages = {“package name”, “package name”}): Scans packages and sub packages from specified packages for Spring Beans. @Configuration: This annotation declares class as a Configuration class which is a source of beans (returns object).  2. @Bean Used over a method in a configuration class which is used to denote that it is returning a bean and the bean is managed by a Spring Container.  3. @Component @Controller @Repository @Service Used at class level. So, no need to declare @Bean over each method. @Repository public interface Repository extends CrudRepository { } @Controller public class StudentController { } @Service public class ServiceImpl implements Service {  } 4. @Autowired Can be used over property, setters, constructors. @Autowired StudentService ss;  5. @Controller • Part of a component. Defines bean. • Used as Presentation layer. • Provides Spring MVC features. • Used if want return view (ex.- .jsp, .html pages)  6. @Service Used at Service Layer.  7. @Repository Used as DAO (Data Access Object) layer. Database Transactions.  8. @Qualifier (Enter bean id) Used if more than one @Bean returns same type of object. Here object will be created for all those objects but only that object Will be returned whose bean id is entered while specifying Qualifier. Ex- @Bean(“Student1”) → @Qualifier(“Student1”)  9. @Lazy Doesn’t create object if the particular method is not called. Object is crated only on request.  10. @RequestMapping( value = “/URL”, method = RequestMethod.GET) • Used over a method to map specific operation URL. @RequestMapping(value = "/update”, method = RequestMethod.PUT)  11. @ResponseBody This tells controller that the object returned is automatically serialized into Jason and passed back as a http response.  12. @RequestBody • Deserializes and converts Jason data into specified object. • Used as a method parameter in a controller. @RequestMapping(value = "/update”, method = RequestMethod.PUT) public String update (@RequestBody Student s)  13. @RestController = @Controller + @ResponseBody • No need to specify @ResponseBody explicitly on individual method. • It will return data into http response (specifies application as webservice) • Comprises properties of both the annotations. • Defined at a class level and defines class as source of bean. • It is a combination of @Controller and @ResponseBody, used for creating a restful controller. It converts the response to JSON or XML. It ensures that data returned by each method will be written straight into the response body instead of returning a template.  14. @PathVariable • Deserializes and converts Jason data into specified object. • Used as a method parameter in a controller to get data from URI. • Need to change mapping according to required path variable. http://localhost:8090/employee/1/john Syntax: - @RequestMapping(value = "/login/{id}/{name}”, method = RequestMethod.GET) public String login(@PathVariable ("id") int id, @PathVariable ("name") int name)  15. @RequestParam • Used when we want to get data from Form. Brings single variable from client side • @RequestParam is a Spring annotation used to bind a web request parameter to a method parameter in query format. http://localhost:8090/employee?id=1&name=john • No need to change mapping. @RequestMapping("/login”) //read the provided form data public String display(@RequestParam("id") String name, @RequestParam("name") String name)  16.@ModelAttribute @ModelAttribute is an annotation that binds a method parameter or method return value to a named model attribute, and then exposes it to a web view. • Brings complete POJO from client side Syntax: - public String saveRegisteredData(@ModelAttribute UserData ud) Note - @ModelAttribute is used for binding data from request param (in key value pairs), but @RequestBody is used for binding data from whole body of the request like POST, PUT request types which contain another format like json, xml.  17.@EnableJPARepository It is used to scan the packages for configuration and repository class for Spring Data JPA.  18.@EntityScan Used to scan all the classes marked with @Entity annotation  19.@Value Used to get value associated with keys in application.properties files. @Value("${spring.mail.username}") private String fromEmail; 20. @CrossOrigin(\*) 21. @Data () Used at class level to reduce boiler-plate code (repetitive code like settergetter etc.). But to use this annotation Lombok dependency is mandatory  **Exception Handling Interview Questions**  What is an exception in java?  In java exception is an object. Exceptions are created when an abnormal situations are arised in our program. Exceptions can be created by JVM or by our application code. All Exception classes are defined in java.lang. In otherwords we can say Exception as run time error.  **State some situations where exceptions may arise in java?**  1) Accesing an element that does not exist in array. 2) Invalid conversion of number to string and string to number. (NumberFormatException) 3) Invalid casting of class (Class cast Exception) 4) Trying to create object for interface or abstract class (Instantiation Exception)  **What is Exception handling in java?**  Exception handling is a mechanism what to do when some abnormal situation arises in program. When an exception is raised in program it leads to termination of program when it is not handled properly. The significance of exception handling comes here in order not to terminate a program abruptly and to continue with the rest of program normally. This can be done with help of Exception handling.  **What is an erorr in Java?**  Error is the subclass of Throwable class in java. When errors are caused by our program we call that as Exception, but some times exceptions are caused due to some environment issues such as running out of memory. In such cases we can’t handle the exceptions. Exceptions which cannot be recovered are called as errors in java. Ex : Out of memory issues.  **What are advantages of Exception handling in java?**  1) Separating normal code from exception handling code to avoid abnormal termination of program. 2) Categorizing in to different types of Exceptions so that rather than handling all exceptions with Exception root class we can handle with specific exceptions. It is recommended to handle exceptions with specific Exception instead of handling with Exception root class. 3) Call stack mechanism : If a method throws an exception and it is not handled immediately, then that exception is propagated or thrown to the caller of that method. This propogation continues till it finds an appropriate exception handler ,if it finds handler it would be handled otherwise program terminates abruptly.  **In how many ways we can do exception handling in java?**  We can handle exceptions in either of the two ways : 1) By specifying try catch block where we can catch the exception. 2) Declaring a method with throws clause . 58) List out five keywords related to Exception handling ? 1) Try 2) Catch 3) throw 4) throws 5) finally.  **Explain try and catch keywords in java?**  In try block we define all exception causing code. In java try and catch forms a unit. A catch block catches the exception thrown by preceding try block. Catch block cannot catch an exception thrown by another try block. If there is no exception causing code in our program or exception is not raised in our code jvm ignores the try catch block. Syntax : try { } Catch(Exception e) { }  **Can we have try block without catch block?**  Each try block requires atleast one catch block or finally block. A try block without catch or finally will result in compiler error. We can skip either of catch or finally block but not both.  **Can we have multiple catch block for a try block?**  In some cases our code may throw more than one exception. In such case we can specify two or more catch clauses, each catch handling different type of exception. When an exception is thrown jvm checks each catch statement in order and the first one which matches the type of exception is execution and remaining catch blocks are skipped. Try with multiple catch blocks is highly recommended in java. If try with multiple catch blocks are present the order of catch blocks is very important and the order should be from child to parent.  **Explain importance of finally block in java?**  Finally block is used for cleaning up of resources such as closing connections, sockets etc. if try block executes with no exceptions then finally is called after try block without executing catch block. If there is exception thrown in try block finally block executes immediately after catch block. If an exception is thrown,finally block will be executed even if the no catch block handles the exception.  **Can we have any code between try and catch blocks?**  We shouldn’t declare any code between try and catch block. Catch block should immediately start after try block. try{ //code } System.out.println(“one line of code”); // illegal catch(Exception e){ // }  **Can we have any code between try and finally blocks?**  We shouldn’t declare any code between try and finally block. finally block should immediately start after catch block.If there is no catch block it should immediately start after try block. try{ //code } System.out.println(“one line of code”); // illegal finally{ // }  **Can we catch more than one exception in single catch block**  From Java 7, we can catch more than one exception with single catch block. This type of handling reduces the code duplication. Note : When we catch more than one exception in single catch block , catch parameter is implicity final. We cannot assign any value to catch parameter. Ex : catch(ArrayIndexOutOfBoundsException || ArithmeticException e) { In the above example e is final we cannot assign any value or modify e in catch statement.  **What are checked Exceptions?**  1) All the subclasses of Throwable class except error,Runtime Exception and its subclasses are checked exceptions. 2) Checked exception should be thrown with keyword throws or should be provided try catch block, else the program would not compile. We do get compilation error. Examples : 1) IOException, 2) SQlException, 3) FileNotFoundException, 4) InvocationTargetException, 5) CloneNotSupportedException 6) ClassNotFoundException 7) InstantiationException  **What are unchecked exceptions in java?**  All subclasses of RuntimeException are called unchecked exceptions. These are unchecked exceptions because compiler does not checks if a method handles or throws exceptions. Program compiles even if we do not catch the exception or throws the exception. If an exception occurs in the program,program terminates . It is difficult to handle these exceptions because there may be many places causing exceptions. Example : 1) Arithmetic Exception 3) ArrayIndexOutOfBoundsException 4) ClassCastException 5) IndexOutOfBoundException 6) NullPointerException 7) NumberFormatException 8) StringIndexOutOfBounds 9) UnsupportedOperationException  **Explain differences between checked and Unchecked exceptions in java?**  Unchecked Exception Checked Exception 1) All the subclasses of RuntimeException are called unchecked exception. All subclasses of Throwable class except RuntimeException are called as checked exceptions 2) Unchecked exceptions need not be handled at compile time Checked Exceptions need to be handled at compile time. 3) These exceptions arise mostly due to coding mistakes in our program. 4) ArrayIndexOutOfBoundsException, ClassCastException, IndexOutOfBoundException SqlException, FileNotFoundException,ClassNotFoundException  **What is default Exception handling in java**  When JVM detects exception causing code, it constructs a new exception handling object by including the following information. 1) Name of Exception 2) Description about the Exception 3) Location of Exception. After creation of object by JVM it checks whether there is exception handling code or not. If there is exception handling code then exception handles and continues the program. If there is no exception handling code JVM give the responsibility of exception handling to default handler and terminates abruptly. Default Exception handler displays description of exception,prints the stacktrace and location of exception and terminates the program. Note : The main disadvantage of this default exception handling is program terminates abruptly.  **Explain throw keyword in java?**  Generally JVM throws the exception and we handle the exceptions by using try catch block. But there are situations where we have to throw userdefined exceptions or runtime exceptions. In such case we use throw keyword to throw exception explicitly. Syntax : throw throwableInstance; Throwable instance must be of type throwable or any of its subclasses. After the throw statement execution stops and subsequent statements are not executed. Once exception object is thrown JVM checks is there any catch block to handle the exception. If not then the next catch statement till it finds the appropriate handler. If appropriate handler is not found ,then default exception handler halts the program and prints the description and location of exception. In general we use throw keyword for throwing userdefined or customized exception.  **Can we write any code after throw statement?**  After throw statement jvm stop execution and subsequent statements are not executed. If we try to write any statement after throw we do get compile time error saying unreachable code.  **Explain importance of throws keyword in java?**  Throws statement is used at the end of method signature to indicate that an exception of a given type may be thrown from the method. The main purpose of throws keyword is to delegate responsibility of exception handling to the caller methods, in the case of checked exception. In the case of unchecked exceptions, it is not required to use throws keyword. We can use throws keyword only for throwable types otherwise compile time error saying incompatible types. An error is unchecked , it is not required to handle by try catch or by throws. Syntax : Class Test{ Public static void main(String args[]) throws IE { } } Note : The method should throw only checked exceptions and subclasses of checked exceptions. It is not recommended to specify exception superclasses in the throws class when the actual exceptions thrown in the method are instances of their subclass.  **Explain the importance of finally over return statement?**  finally block is more important than return statement when both are present in a program. For example if there is any return statement present inside try or catch block , and finally block is also present first finally statement will be executed and then return statement will be considered.  **Explain a situation where finally block will not be executed?**  Finally block will not be executed whenever jvm shutdowns. If we use system.exit(0) in try statement finally block if present will not be executed.  **Can we use catch statement for checked exceptions?**  If there is no chance of raising an exception in our code then we can’t declare catch block for handling checked exceptions .This raises compile time error if we try to handle checked exceptions when there is no possibility of causing exception.  **What are user defined exceptions?**  To create customized error messages we use userdefined exceptions. We can create user defined exceptions as checked or unchecked exceptions. We can create user defined exceptions that extend Exception class or subclasses of checked exceptions so that userdefined exception becomes checked. Userdefined exceptions can extend RuntimeException to create userdefined unchecked exceptions. Note : It is recommended to keep our customized exception class as unchecked,i.e we need to extend Runtime Exception class but not Excpetion class.  **Can we rethrow the same exception from catch handler?**  Yes we can rethrow the same exception from our catch handler. If we want to rethrow checked exception from a catch block we need to declare that exception.  **Can we nested try statements in java?**  Yes try statements can be nested. We can declare try statements inside the block of another try statement.  **Explain the importance of throwable class and its methods?**  Throwable class is the root class for Exceptions. All exceptions are derived from this throwable class. The two main subclasses of Throwable are Exception and Error. The three methods defined in throwable class are : 1) void printStackTrace() : This prints the exception information in the following format : Name of the exception, description followed by stack trace. 2) getMessage() This method prints only the description of Exception. 3) toString(): It prints the name and description of Exception.  **Explain when ClassNotFoundException will be raised ?**  When JVM tries to load a class by its string name, and couldn’t able to find the class classNotFoundException will be thrown. An example for this exception is when class name is misspelled and when we try to load the class by string name hence class cannot be found which raises ClassNotFoundException.  **Explain when NoClassDefFoundError will be raised ?**  This error is thrown when JVM tries to load the class but no definition for that class is found NoClassDefFoundError will occur. The class may exist at compile time but unable to find at runtime. This might be due to misspelled classname at command line, or classpath is not specified properly , or the class file with byte code is no longer available  **Java 8 features :**  There is list of below features of java 1.8 as  Functional interface  Lambda Expression  Default method  Static method  For each Method  Optional Class  String Joiners  **Functional Interface**  An Interface that contains exactly one abstract method is known as functional interface.  It can have any number of default, static methods but can contain only one abstract method.  Example-  **package** com.test;  @FunctionalInterface  **public** **interface** Test {  **void** getStudentName(String name);    }  **package** com.test;  **public** **class** Main **implements** Test {  @Override  **public** **void** getStudentName(String name) {  System.***out***.println(name);  }  **public** **static** **void** main(String[] args) {  Main main = **new** Main();  main.getStudentName("ashok");  }  }  **Lambda Expression-**  Why?  Less coding  Syntax- (argument-list) -> {body}     * **Argument-list:** It can be empty or non-empty as well. * **Arrow-token:** It is used to link arguments-list and body of expression. * **Body:** It contains expressions and statements for lambda expression.   **No Parameter Syntax**  () -> {  //Body of no parameter lambda  }  **One Parameter Syntax**  (p1) -> {  //Body of single parameter lambda  }  **Two Parameter Syntax**  (p1,p2) -> {  //Body of multiple parameter lambda  }  Example-  **package** com.test;  **public** **interface** Addition {  **int** add(**int** a,**int** b);  }  **package** com.test;  **public** **class** Main {  **public** **static** **void** main(String[] args) {  // Multiple parameters in lambda expression  Addition addition = (a, b) -> (a + b);  System.***out***.println(addition.add(10, 20));  // Multiple parameters with data type in lambda expression  Addition addition2 = (**int** a, **int** b) -> (a + b);  System.***out***.println(addition2.add(100, 200));  }  }  Output  30  300  **Default method**  Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default are known as default methods. These methods are non-abstract methods.  Example-  **package** com.test;  **public** **interface** Example {  **default** **void** m1() {  System.***out***.println("this is default m1 method");  }  }  **package** com.test;  **public** **class** TestMain **implements** Example {  **public** **static** **void** main(String[] args) {    TestMain testMain=**new** TestMain();  testMain.m1();  }  }  Output  this is default m1 method  **Static method-**  Java provides a facility to create static methods inside the interface.  **package** com.demo;  **public** **interface** Example {  **static** **void** x1() {  System.***out***.println("this is static method");  }  }  **package** com.demo;  **public** **class** MainTest **implements** Example{  **public** **static** **void** main(String[] args) {    Example.*x1*();  }  }  **forEach () method-**  The Java forEach() method is a utility function to iterate over a collection such as (list, set or map) and [stream](https://howtodoinjava.com/java8/java-streams-by-examples/). It is used to perform a given action on each the element of the collection.  **package** com.test;  **import** java.util.HashMap;  **import** java.util.Map;  **public** **class** MapDemo {  **public** **static** **void** main(String[] args) {  Map<String, String> map = **new** HashMap<String, String>();  map.put("10", "ram");  map.put("11", "shyam");  map.put("12", "ganesh");  map.forEach((k, v) -> System.***out***.println("Key = " + k + ", Value = " + v));  }  }  Output  Key = 11, Value = shyam  Key = 12, Value = ganesh  Key = 10, Value = ram  **Optional class-**  Java introduced a new class Optional in jdk8. It is a public final class and used to deal with NullPointerException in Java application.  You must import java.util package to use this class. It provides methods which are used to check the presence of value for particular variable.  Why?  **package** com.test;  **public** **class** MapDemo {  **public** **static** **void** main(String[] args) {  String[] str = **new** String[10];  String lowercaseString = str[5].toLowerCase();  System.***out***.print(lowercaseString);  }  }  Exception in thread "main" java.lang.NullPointerException  at com.test.MapDemo.main(MapDemo.java:8)  Here we are getting exception, to avoid this type of exception, we should go for optional class  **package** com.test;  **import** java.util.Optional;  **public** **class** MapDemo {  **public** **static** **void** main(String[] args) {  String[] str = **new** String[10];  Optional<String> checkNull = Optional.*ofNullable*(str[5]);  **if** (checkNull.isPresent()) { // check for value is present or not  String lowercaseString = str[5].toLowerCase();  System.***out***.print(lowercaseString);  } **else**  System.***out***.println("string value is not present");  }  }  Output  string value is not present.  **Java String Joiner-**  Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by a delimiter. Now, you can create string by passing delimiters like comma(,), hyphen(-) etc  Example  **import** java.util.StringJoiner;  **public** **class** Example {  **public** **static** **void** main(String[] args) {  StringJoiner stringJoiner = **new** StringJoiner(","); // passing comma(,) as delimiter  // Adding values to StringJoiner  stringJoiner.add("Ram");  stringJoiner.add("Shyam");  stringJoiner.add("ashok");  stringJoiner.add("ajay");  System.***out***.println(stringJoiner);  }  }  Output  Ram,Shyam,ashok,ajay  **Hibernate :**  Hibernate frameworks is mediator through which java application is communicated with database. It is open source frameworks. It is Object Relational Mapping (ORM) tool.  **Why?**   * In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections. * We must close the connection, or we may get a chance to get connections error message. * Actually if we didn’t close the connection in the finally block, then jdbc doesn’t responsible to close that connection. * In JDBC we need to write Sql commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious. * To overcome above drawbacks we should go for Hibernate framework.   **Advantages of Hibernate-**   1. It is open source frameworks. 2. Faster performance-                It uses cache concept hence the performance is fast.   1. Database independent query-                It generates the database independent query.   1. Automatic table creation-                It has facility to create the database tables automatically. There is no need to create the database tables manually.   1. Simplifies the complex join-                It is easy to fetch the data from multiple tables in hibernate framework.  **Hibernate Cache Support-**  **Why?**  To reduce the number of calls from database and improve the application performance.  **How it works?**              It will fetch only one time data from database and store it on local cache (temporary storage).               For n no. of user retrieve data only once from database and store it to cache and return it to user.  There are **three types** of cache in the hibernate as  Session level cache  SessionFactory level cache  Query level cache  Problem without cache-              Suppose I have Java study material application or site. It is the constant data, suppose 1 lakh user daily visit to site to read the concepts, so it will hit 1 lakh times  to database due to this your application will slow and lot of processing time it will takes. To overcome this issue, we should go for cache supports.  **Session level cache-**  It is called as first level cache.  It is apply for single user or one user only.  It is default level cache.  The first level cache data will not be available to entire application because application can use many session objects.  **When to use?**  Example- Login to Gmail application, if you want to retrieve the inbox mails at first time login. It will load the data from database. If you trying to refresh and if you do not have new mails. The data instead of reading every time from database, it will load from cache itself until doing logout. It will do only one select operation.  Note-  It will fetch data only one time from database and store it on session objects next time when user request some data, so it will retrieve data from session objects.  **Query level cache-**  **Why?**        If I want to execute some query again and again or multiple times (Example select max salary from employee) then you should go for query level cache.  **SessionFactory level cache-**  It is called as second level cache.  It will apply for all the users.  Program-2 Using SessionFactory  **Why?**  Problem with session level cache  https://lh6.googleusercontent.com/GXh0Odbo2osxXGz9TF02-ACi5-_OjuPRdQKAznBNCfg4pnwhNlITDf7JrzLZFM7xKPRShcdU4NUrn7PHnZ1M3e68PspGQTXUhgeWa-6V1JnG8o-QF6yLYXMzcrMgeJkHBCg3DLZmaznE4mBxppwidHnbQiT9V3F2PxmNCw1nk7jXs9RRNTD6wM2co7275fn-LW1QCg              Here admin1 and admin2 are the trying to access the same data, if we are using session, it will create the two session objects for some data or common data but if we use sessionfactory cache, data will fetch from database only one(data store under the sessionfactory) and data will available to any user.  **When?**  It is useful if you have multiple session objects from sessionfactory. |  |