**Q1 : What is Inheritance and Benefit of Inheritance?**

The main **advantages of inheritance** are code reusability and readability. When child class **inherits** the properties and functionality of parent class, we need not to write the same code again in child class. This makes it easier to reuse the code, makes us write the less code and the code becomes much more readable.

**Q2: What is difference between IS-A and Composition Has-A Relationship?**

|  |  |
| --- | --- |
| **IS-A** | **HAS-A** |
| It isÂ known as an inheritance | Has-A relationship is also known as Composition and Aggregation |
| The main advantage of IS-A relationship is code reusability.  By default parent properties available | Parent properties not available |
| By using extends keyword we can implement IS-A relationship. | There is no specific keyword to implement HAS-A relation  but most of the time we are depending on the new keyword. |

**Q3: What is association in JAVA?**

Association in Java is a connection between two separate classes that is set up through their objects. Although, Java association can balance, one-to-one, one-to-many, and many-to-many relationships. It defines the multiplicity between objects.

There are two types of Association:

1. Aggregation(Bike🡪Engine)or(Bank🡪Employee)
2. Composition(College🡪Department🡪Student)

**Q4: What is polymorphism in java?**

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

**In Java polymorphism is mainly divided into two types:**

* Compile time Polymorphism
* Runtime Polymorphism

1. **Compile-time polymorphism**: It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading. But **Java doesn’t support the Operator Overloading**.
2. [**Runtime polymorphism**](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/)**:** It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

**Q5: what is method overloading in java?**

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

**Q6: What is method overriding in java?**

If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.( Method overriding is used for runtime polymorphism)

**Q7: Rules for Method Overriding?**

* The method must have the same name as in the parent class.
* The method must have the same parameter as in the parent class.
* public, protected and default (in the same package) can be overridden.
* Final and static methods cannot be overridden.
* The overriding method must have same argument list.
* The overriding method must have same return type (or subtype).
* The overriding method must not have more restrictive access modifier.

1. default--> default, protected or public.
2. protected->protected or public.
3. public-->public

* The overriding method must not throw new or broader checked exceptions.
* Constructors cannot be overridden.

**Q8: What is Abstraction in Java?**

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Shape🡪drow()🡪 Circle implements shape🡪 Shape shape = new Circle();🡪 shape.drow();

**Q9: What is encapsulation in Java?**

The process of binding data and corresponding methods (behavior) together into a single unit is called **encapsulation in Java**.

School Bag🡪pen, pencil ,notebook.

**Q10: Differences between Interface and Abstract Class?**

|  |  |  |
| --- | --- | --- |
| **Type** | **Interface** | **Abstract class** |
| Type of method | Only abstract methods. From Java 8, it can have default and static methods also. | abstract and non-abstract methods |
| Final Variables | default final | may contain non-final variables |
| Type of variables | static and final variables. | final, non-final, static and non-static |
| Implementation | Interface can’t provide the implementation of abstract class | Abstract class implementation of interface |
| Inheritance | Supports multiple inheritance. | Doesn’t support multiple inheritance. |

**Q11: Difference between Abstraction and Encapsulation.**

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| Abstraction is the process or method of gaining the information | While encapsulation is the process or method to contain the information. |
| Abstraction is the method of hiding the unwanted information | Whereas encapsulation is a method to hide the data in a single entity  or unit along with a method to protect information from outside |
| We can implement abstraction using abstract class and interfaces | Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public. |
| In abstraction, implementation complexities are hidden using abstract classes and interfaces | While in encapsulation, the data is hidden using methods of getters and setters. |

**Q12: what is clone in java?**

The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**

**Q13. Types of clones in java?**

Java supports two type of cloning: - **Deep and shallow cloning.** By default shallow clone is used in Java. Object class has a method clone() which does shallow cloning

**Q14. What is Shallow copy?**

New object is pointing to the same memory reference of the old object. The memory usage is lower.

**Q15:What is deep copy and how it can be achieved?**

A new memory is allocated for the object and contents are copied.

**Q16) What is difference between deep and shallow cloning?**

shallow copying of this object will be pointing to the same memory reference as the original object. So a change in myData by either original or cloned object will be reflected in other also. But in deep copying there will memory allocated and values assigned to the property will be same. Any change in object will not be reflected in other.

**Q17: What is difference between Object class equals method and String class equals method?**

* Object class equals() method in Object class used to compare whether some other Object is "equal to" this one. It compares only the Object level are same in both the sides.
* Basically equals() method in String class is overridden from Object class and used to compare whether some other Object is "equal to" this one, along with also Objects will be typecast to String and each character wise comparison will be made in equals() method

**Q18) How memory Management Done in Java?**

**JVM:**

* + JVM is a Runtime engine to run java based applications.
  + JVM is part of JRE which is the part of JDK.
  + JVM is responsible for **load** and **run** java code line by line.

**Class Loader Sub-system:**

To Load dot class file in the memory of JVM

**Memory Area**

Method Area

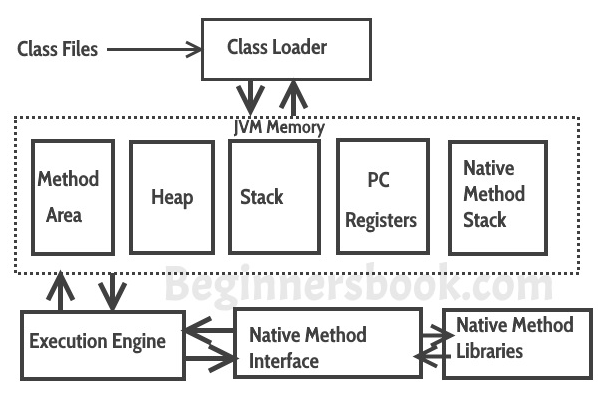
Heap Memory

Stack Memory

PC registers

Native method stacks

**Basic Top Level JVM architecture**



**Class Loader Sub-system:**

To Load dot class file in the memory of JVM and it Responsible for below activities

* 1. **Loading**
  2. **Linking**
  3. **Initialization**

**Loading:**

Reading dot class file information and store corresponding binary information into method area memory. (All data related to class like parent, fully qualified name, constants, variable etc...)

After loading dot class file Immediately JVM will creates an Object of the type Class to represent the corresponding class’s binary information on the Heap memory.

Ex:-

Class c = Class.forName(“Employee”);

Method[] m =c.getDeclaredMethods();

Field[] f=c.getDeclaredFields();  
//-------------------------------------------------

Employee e1 = new Employee();

Class c1 = e1.getClass();

Employee e2 = new Employee();

Class c2 = e2.getClass();

C1==c2🡪true and hashcode is same

**Linking:**

Performing below 3 activities

* **Verification**

It is process of ensuring that binary representation of class is structurally correct or not.

i.e. weather that class file properly formatted or not or it compiled by valid compiler or not(Byte code Verifier🡪 VerifyError exception)

* **Preparation**

Allocate memory for static variable and assign default values.

* **Resolution**

It is the process of replacing all Symbolic references used in class with original references from method area

For every class JVM is maintaining one constant pool for storing the symbols.

Symbolic representation form method area replaced with machine language symbolic representation.

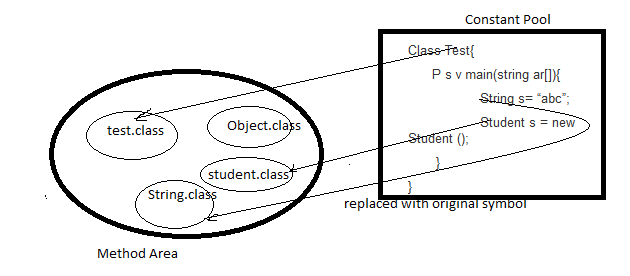
Class Test{

P s v main(string ar[]){

String s= “abc”;

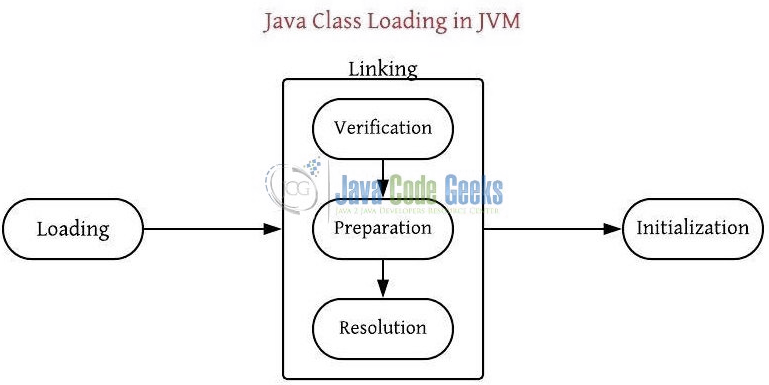
Student s = new Student ();

}

} 

**Initialization**

For static variables original value assign and static blocks will execution



**Loading:**

* **Bootstrap Class Loader**

To Load Classes from bootstrap class path (rt.jar)

Jdk🡪jre🡪lib🡪rt.jar(implemented in native languages)

* **Extension Class Loader**(child of bootstrap class loader)

To Load Classes from extension path

Jdk🡪jre🡪lib🡪ext🡪any jar and .class file

* **Application Class Loader**(child of extension class loader)

To load classes form application path(AppclassLoader)

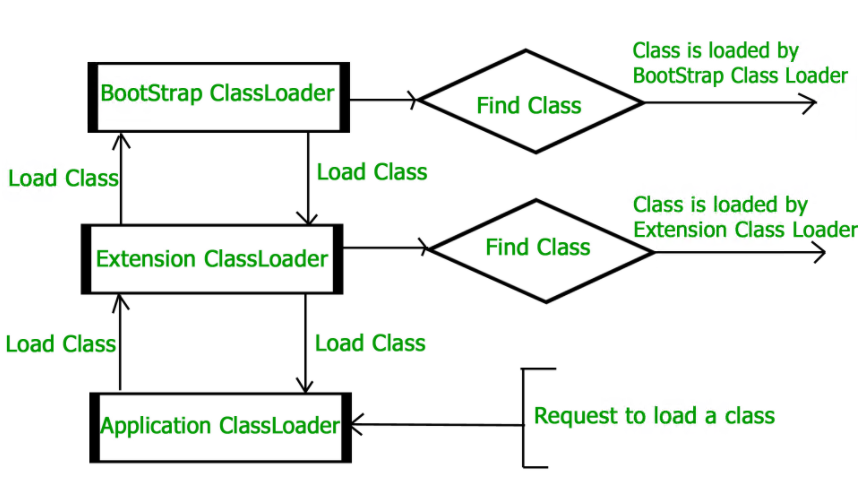
**Delegation Hierarchy algorithm is USED**

Ex:

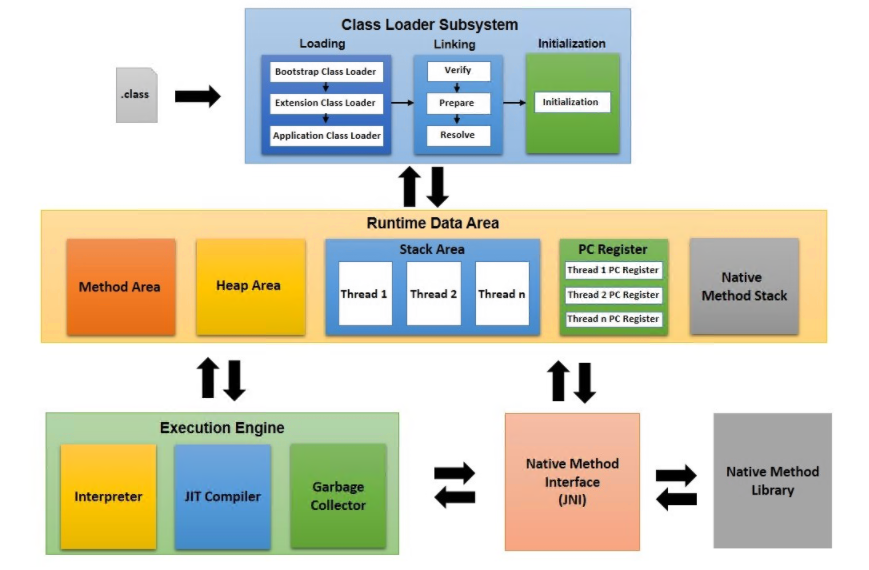
String.class.getClassLoade();//null

Student.class.getClassLoade();//(ext dir) ExtClassLoadder

Test.class.getClassLoade();//(application dir) AppClassLoadder

****

**Complete Architecture of JVM**

****

**Method Area:**

Class level data stored and static variable

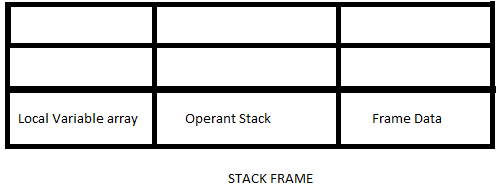
**Heap Memory:**

Object and corresponding instance Variables.

**Stack Memory:**

For every thread a separate run time stack will be created.

Each and every method call performed by in stack memory



**PC Registers:**

For every thread a Separate pc registers, to hold the address of next executing instruction.

**Native Method Stack:**

For every thread a separate stack will be created, to hold the native method information.

**Execution Engine:**

Execution engine is central part of JVM to execute the byte code which generated by compiler

**Interpreter**:

Interpreter is responsible to read interpret and execute java program line by line.

**JIT:**

If any method repeatedly required method

**Profiler🡪** finding hotspots (method repeatedly required method)

**JAVA NATIVE INTERFACE🡪** if any native required method provide

**Q19) what is immutable class in java ? List of predefined classes in java?**

* Immutable class means that once an object is created, we cannot change its content.
* In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable. We can create our own immutable class as well.

Q) **Why we required Immutable classes in Java?** An immutable class is **good for caching purposes because you don't have to worry about the value changes**. An immutable class is inherently **thread-safe**, so you don't have to worry about thread safety in multi-threaded environment.

**Q20) How you can write custom immutable class with example?**

1. Make your class final, so that no other classes can extend it.
2. Make all your fields final, so that they’re initialized only once inside the constructor and never modified afterward.
3. Don’t expose setter methods.
4. When exposing methods which modify the state of the class, you must always return a new instance of the class.
5. If the class holds a mutable object:

* Make sure to always return a clone copy of the field and never return the real object instance.

1. Example

**public** **class** Age {

**private** **int** day;

**private** **int** month;

**private** **int** year;

**public** **int** getDay() {

**return** day;

}

**public** **void** setDay(**int** day) {

**this**.day = day;

}

**public** **int** getMonth() {

**return** month;

}

**public** **void** setMonth(**int** month) {

**this**.month = month;

}

**public** **int** getYear() {

**return** year;

}

**public** **void** setYear(**int** year) {

**this**.year = year;

}

}

**public** **final** **class** ImmutableStudent {

**private** **final** **int** id;

**private** **final** String name;

**private** Age age;

**public** **int** getId() {**return** id; }

**public** String getName() { **return** name; }

**public** Age getAge() {

Age cloneAge = **new** Age();

cloneAge.setDay(age.getDay());

cloneAge.setMonth(age.getMonth());

cloneAge.setYear(age.getYear());

**this**.age =cloneAge;

**return** age;

//return age; //change the address or it effeted;

}

**public** ImmutableStudent(**int** id, String name, Age age) {

**this**.id = id; **this**.name = name; **this**.age = age;

}

**public** **static** **void** main(String[] args) {

Age age = **new** Age();

age.setDay(1);

age.setMonth(1);

age.setYear(1992);

ImmutableStudent student = **new** ImmutableStudent(1, "Alex", age);

System.***out***.println("Alex age year before modification = " + student.getAge().getYear());

age.setYear(1995);

System.***out***.println("Alex age year before modification = " + student.getAge().getYear());

}

}

***Collection***

**Q21) what is difference between List, Set and Map?**

|  |  |  |
| --- | --- | --- |
| **List** | **Set** | **Map** |
| The list interface allows duplicate elements | Set does not allow duplicate elements | The map does not allow duplicate elements |
| The list maintains insertion order | Set do not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in set almost only one null value. | The map allows a single null key at most and any number of null values |
| List implementation classes are Array List  LinkedList. | Set implementation classes are HashSet, LinkedHashSet, and TreeSet. | Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap. |
| The list provides get() method to get the element at a specified index. | Set does not provide get method to get the elements at a specified index | The map does not provide get method to get the elements at a specified index |
| If you need to access the elements  frequently by using the index then we can use the list | If you want to create a collection of unique elements then we can use set | If you want to store the data in the form  of key/value pair then we can use the map. |
| To traverse the list elements by using Listlterator. | Iterator can be used traverse the set elements | Through keyset, value, and entry set. |

**Q22) what is difference between Array List and Linked List?**

|  |  |
| --- | --- |
| **Array List** | **Linked List** |
| ArrayList internally uses a dynamic array to store the elements. | LinkedList internally uses a doubly linked list to store the elements. |
| Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| An ArrayList class can act as a list only because it implements List only. | LinkedList class can act as a list and queue both because it implements List and Deque interfaces. |
| ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |

**Q23) what is difference between ArrayList and LinkedList?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code | Hashtable is synchronized. It is thread-safe and can be shared with many threads. |
| HashMap allows one null key and multiple null values. | Hashtable doesn't allow any null key or value. |
| HashMap is fast. | Hashtable is slow. |
| HashMap is traversed by Iterator. | Hashtable is traversed by Enumerator and Iterator. |
| Iterator in HashMap is fail-fast. | Enumerator in Hashtable is not fail-fast |
| HashMap inherits AbstractMap class. | Hashtable inherits Dictionary class. |

**Q24) what is difference between HashSet, TreeSet and LinkedHashSet?**

|  |  |  |
| --- | --- | --- |
| **HashSet** | **TreeSet** | **LinkedHashSet** |
| HashSet is fastest than LinkedHashSet and TreeSet | TreeSet is slow when compared with both Hashset and LinkedHashSet | LinkedHashSet is second fastest next to HashSet |
| HashSet does not maintain any order | TreeSet maintains Sorting Order | LinkedHashSet maintains insertion order |
| HashSet allows null | TreeSet does not allow null | LinkedHashSet allows null |
| HashSet uses equals() method | TreeSet uses compareTo() method | LinkedHashSet uses equals() method |
| HashSet backed by HashMap | TreeSet backed by NavigableMap | LinkedHashSet backed by HashSet |

**Q25) what is difference between HashMap, TreeMap and LinkedHashMap?**

|  |  |  |
| --- | --- | --- |
| **HashMap** | **TreeMap** | **LinkedHashMap** |
| HashMap has complexity of O(1) for insertion and lookup. | TreeMap has complexity of O(logN) for insertion and lookup. | LinkedHashMap has complexity of O(1) for insertion and lookup. |
| HashMap allows one null key and multiple null values. | TreeMap does not allow null key but allow multiple null values. | LinkedHashMap allows one null key and multiple null values. |
| HashMap does not maintain any order. | TreeMap maintains order. It stores keys in sorted and ascending order. | LinkedHashMap maintains order in which key-value pairs are inserted. |

**Q26) what is difference between Vector and List?**

|  |  |
| --- | --- |
| **Vector** | **List** |
| It has contiguous memory. | While it has non-contiguous memory. |
| It is synchronized. | While it is not synchronized. |
| Vector may have a default size. | List does not have default size. |
| In vector, each element only requires the space for itself only. | In list, each element requires extra space for the node which holds the element, including pointers to the next and previous elements in the list. |
| Insertion at the end requires constant time but insertion elsewhere is costly. | Insertion is cheap no matter where in the list it occurs. |
| Vector is thread safe. | List is not thread safe. |
| Deletion at the end of the vector needs constant time but for the rest it is O(n). | Deletion is cheap no matter where in the list it occurs. |
| Random access of elements is possible. | Random access of elements is not possible. |
| Iterators become invalid if elements are added to or removed from the vector. | Iterators are valid if elements are added to or removed from the list. |

**Q27) what is difference between ArrayList and Vector?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is not synchronized. | Vector is synchronized. |
| ArrayList increments 50% of current array size if the number of elements exceeds from its capacity. | Vector increments 100% means doubles the array size if the total number of elements exceeds than its capacity. |
| ArrayList is fast because it is non-synchronized. | Vector is slow because it is synchronized, |

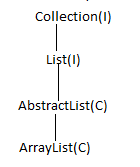
**Q28) what is difference between Comparator and Comparable?**

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

**Q35 Difference between ClassNotFoundException and NoClassDefFoundError?**

|  |  |
| --- | --- |
| **ClassNotFoundException** | **NoClassDefFoundError** |
| It is an exception. It is of type java.lang.Exception. | It is an error. It is of type java.lang.Error. |
| It occurs when an application tries to load a class at run time which is not updated in the classpath. | It occurs when java runtime system doesn’t find a class definition, which is present at compile time, but missing at run time. |
| It is thrown by the application itself.  It is thrown by the methods like  Class.forName(), loadClass() and findSystemClass(). | It is thrown by the Java Runtime System. |
| It occurs when classpath is not updated with required JAR files. | It occurs when required class definition is missing at runtime. |

**Q29 Internal Working of ArrayList:**

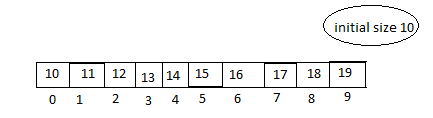
****

In list allow duplicate element and it displayed same as inserted order (insertion or order is pre reserved)

Java 7

ArrayList list= new ArrayList();

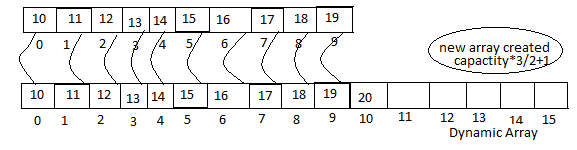
List.add(10); List.add(11); List.add(12);............... List.add(19);



List.add(20);

New array size =Current capacity \*3/2 +1

10\*3/2+1=16



Java8

When First element is inserted then initial capacity will be 10

NewCapacity = oldCapacity+(oldCapacity>>1)🡪50% increased

Once new array is created older array will be eligible for garbage collector.

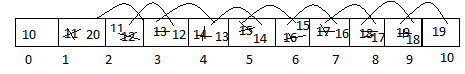
For Duplicate element added in list

List.add(10); List.add(11); List.add(12); List.add(13); List.add(13); List.add(13); List.add(null);



Insertion and middle level shifting operation is performed

List.add(1,20);



List.remove(6);

For at middle deletion also shifting happen.

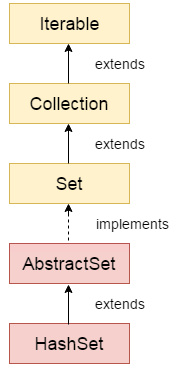
For retrieve data form list it is too fast because random access interface

Object o =List.get(0);//10🡪 1 sec

Object o =List.get(4);//13🡪 1 sec

Object o =List.get(1000);//13🡪 1 sec

**Q30 Internal Working of HashSet and HashMap**

****

HashSet is an implementation class of Set as we know set contains only unique elements.

HashSet hs = new HashSet();

Public HashSet(){

Map = new HashMap();

}

Hs.add(“abc”);

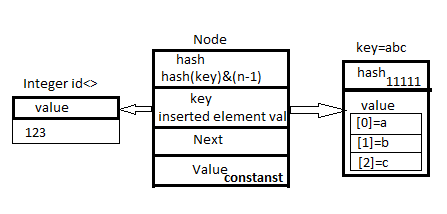
Hs.add(123);

Public boolean add(Object E){

return Map.put(E,PRESENT)==null;

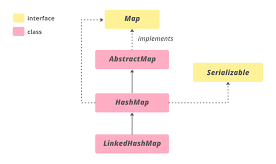
}

HashMap will be created and when first element is inserted then default capacity is 16



If two keys hash code is same then hash Collision will be occurred an data will be stored in next (create another note)

**HASHMAP:**

****

Hashing is a process of converting an object into integer form by using the method hashCode()

Index calculations

HashMap contains an array of Node and Node can represent a class having following objects :

* int hash
* K key
* V value
* Node next

If hash map reaches to thresholds then size will increased using below formula

* capacity = number of buckets \* load factor

HashMap map = new HashMap();

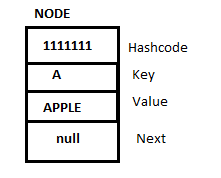
Put method is called 16 array of buckets then created and assign default values is null except 1st element

Map.put(‘A’,”APPLE”);{

Int hashCode = hashCode(key);//111111111

Int idex = hash(key)&(size+1)//4

}





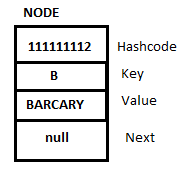
Map.put(‘B’,”BARCARY”);{

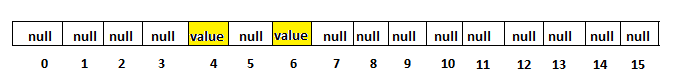
Int hashCode = hashCode(key);//111111112

Int idex = hash(key)&(size+1)//6

}

HashMap size is=1





HashMap size is=2

Duplicate Key insertion with new values

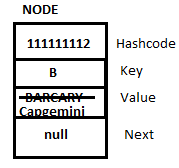
Map.put(‘B’,”Capgemini”);{

Int hashCode = hashCode(key);//111111112

Int idex = hash(key)&(size+1)//6

}

Only value will be updated in current node and size will be same



In above condition hash code and key is same .equals method will not be call.

**Hash Collision:**

When in hashmap hash code is same but key is different then hash collision occourd.

Map.put(‘EA’,”INFOSYS”);{

int hashCode = hashCode(key);//2208

Int index = hash(key)&(size+1)//6

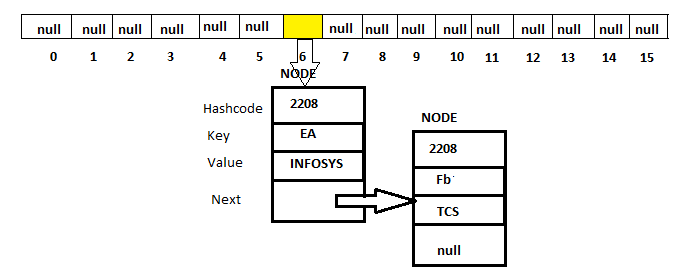
}

Map.put(‘Fb’,”TCS”);{

int hashCode = hashCode(key);//2208

int index = hash(key)&(size+1)//6

}



In hash collision .equals method is called and it returns always false and element inserted and old nodes next is pointing to new node.

If there 5 elements in next🡪next node then .equals method called 5 times then it return false and node is inserted

**Get Method Operations in HashMap**:

map.get(“EA”){

int hashCode = hashCode(key);//2208

int index = hash(key)&(size+1)//6

}

Depending on index return the value.

**Hash Collision at get Method():**

map.get(“Eb”){

int hashCode = hashCode(key);//2208

int index = hash(key)&(size+1)//6

}

When same hash code and same index is calculated then .equal method called if key matched return values if not matched then it return false. (dot equals method call for all next element present in Node upto element found)

**Q31 What id general Contract between hashCode() and equals():**

If two objects are equal(according to **equals()** method) then the **hashCode()** method should return the same integer value for both the objects.

But, it is not necessary that **the hashCode()** method will return the distinct result for the objects that are not equal (according to **equals()** method).

**Q32 How to Covert Hashmap into ArrayList?**

Set<Entry<Integer, String>> entry = hashMap.entrySet();

ArrayList<Entry<Integer, String>> asList =

**new** ArrayList<Entry<Integer, String>>(entry);

**Q33 how many ways we create object in java**

1. Using new Keyword
2. Using newInstance() of Class class
3. Using [clone()](https://www.geeksforgeeks.org/clone-method-in-java-2/) of Cloneable interface
4. Using Serialization  and Deserialization Object
5. Using Factory Method

**Q34 How to Sort array list in custom property?**

Java 7🡪Collections.sort(list, new Mycomparator());

Java8 🡪List<Integer> asList = al.stream().sorted((a,b)-> a<b?1:a>b?-

1:0).collect(Collectors.toList());

**Java8**

**What is Stream?**

Stream represents a sequence of objects from a source, which supports aggregate operations

* Sequence of elements
  + Set of elements of specific type in a sequential manner
  + A stream gets/computes elements on demand
* Source
  + Stream takes Collections, Arrays as input source
* Aggregate operations
  + Stream supports aggregate operations like filter, map, and limit, reduce, find, and match, and so on.
* Pipelining
  + Most of the stream operations return stream itself so that their result can be pipelined.
  + These operations are called intermediate operations and their function is to take input, process them, and return output to the target
  + collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.
* Automatic iterations
  + Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.
* **forEach**
  + to iterate each element of the stream
* **Map**
  + The ‘map’ method is used to map each element perform some operation.
* **Filter**
  + The ‘filter’ method is used to eliminate elements based on a criteria

**Q36 Difference between Function and Procedure?**

|  |  |  |
| --- | --- | --- |
| **Key** | **Function** | **Procedure** |
| Definition | A function is used to calculate result using given inputs. | A procedure is used to perform certain task in order. |
| Call | A function can be called by a procedure. | A procedure cannot be called by a function. |
| DML | DML statments cannot be executed within a function. | DML statements can be executed within a procedure. |
| SQL, Query | A function can be called within a query. | A procedure cannot be called within a query. |
| SQL, Call | Whenever a function is called, it is first compiled before being called | A procedure is compiled once and can be called multiple times without being compiled. |
| SELECT | A select statement can have a function call. | A select statemnt can't have a procedure call. |

Q1)How to integrate hibernate with Spring?

Add below properties in spring\_servlet.xml file

BasicDataSource

<data base properties>

LocalSessionFactoryBean

<mappingResources,hibernateProperties>

HibernateTemplate

<sessionFactory>

Q2)Which annotations used in Spring MVC?

* @Required

This annotation is applied on bean setter methods. Consider a scenario where you need to enforce a required property

* @Autowired

This annotation is applied on fields, setter methods, and constructors. The @Autowired

 annotation injects object dependency implicitly.

* @Qualifier:

This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used. @Qualifier can be specified on individual constructor arguments or method parameters. This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.

* @Configuration

This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.

* @Bean

This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans. As mentioned earlier, @Configuration will have methods to instantiate and configure dependencies. Such methods will be annotated with @Bean. The method annotated with this annotation works as bean ID and it creates and returns the actual bean.

@Bean

public Address address(){

return new Address();

}

* @Value

This annotation is used at the field, constructor parameter, and method parameter level. The @Value annotation indicates a default value expression for the field or parameter to initialize the property with

**Spring Framework Stereotype Annotations**

* @Component

This annotation is used on classes to indicate a Spring component. The @Component annotation marks the Java class as a bean or say component so that the component-scanning mechanism of Spring can add into the application context.

* @Service

This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs.

* @Repository

This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.

**Spring MVC and REST Annotations.**

* @Controller

This annotation is used on Java classes that play the role of controller in your application.

* @RequestMapping

This annotation is used both at class and method level. The @RequestMapping annotation is used to map web requests onto specific handler classes and handler methods.

* @ExceptionHandler.

This annotation is used at method levels to handle exception at the controller level.

* @PathVariable:

Read the values form URI

* @RequestBody

This annotation is used to annotate request handler method arguments. The @RequestBody annotation indicates that a method parameter should be bound to the value of the HTTP request body.

* @RequestParam

Read Request parameter values.

* @ResponseBody

The @ResponseBody annotation indicates that the result type should be written straight in the response body in whatever format you specify like JSON or XML. Spring converts the returned object into a response body by using the HttpMessageConveter.

* @ControllerAdvice

This annotation is applied at the class level. As explained earlier, for each controller you can use @ExceptionHandler on a method that will be called when a given exception occurs. But this handles only those exception that occur within the controller in which it is define.

* @RestController
  + This annotation is used at the class level. The @RestController annotation marks the class as a controller where every method returns a domain object instead of a view.
  + By annotating a class with this annotation you no longer need to add @ResponseBody.