1. **Q. Why we use exchange method rest template in spring boot ?**

Ans .

The exchange method executes the request of any HTTP method and returns ResponseEntity instance. The exchange method can be used for HTTP DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, TRACE methods. Using exchange method we can perform CRUD operation i.e. create, read, update and delete data. The exchange method returns ResponseEntity using which we can get response status, body and headers.

2**. Q. what do you know about @Data annotation?**

Ans : @Data annotation is an “all -in one” annotation group of multiple annotation

@toString

@Getter

@Setter

@EqualsAndHashCode

@RequiredArgConstructor

3. **Q. Fail fast vs fail-safe ?**

Ans:

Fail Fast :

Fail-fast iterators in Java don't play along when the underlying collection gets modified.

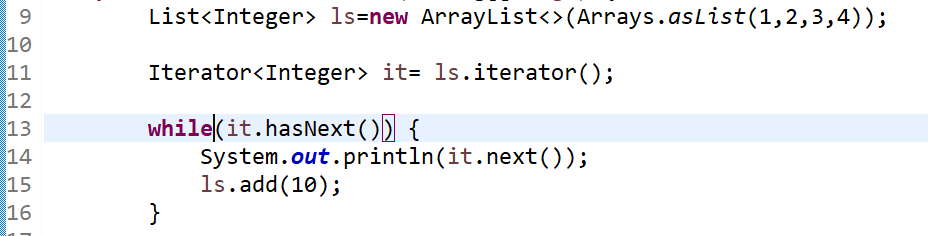
Collections maintain an internal counter called modCount. Each time an item is added or removed from the Collection, this counter gets incremented.

When iterating, on each next() call, the current value of **modCount** gets compared with the initial value. If there's a mismatch, it throws **ConcurrentModificationException** which aborts the entire operation.

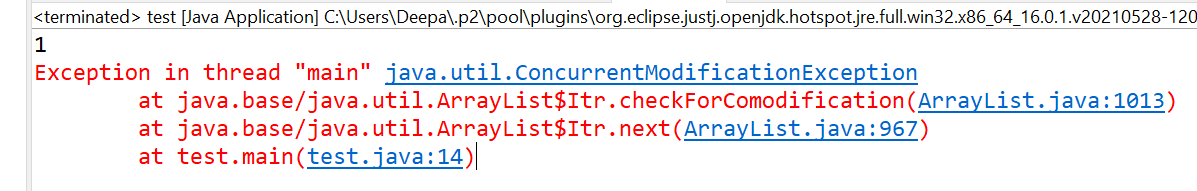
If during iteration over a Collection, **an item is removed using Iterator‘s remove() method, that's entirely safe and doesn't throw an exception**.

**Collection like:**  ArrayList , HashMap

**Code Example :**



Output:



Explanation :

In second iteration it will throw an error on line number 14. Due to in first iteration we modify the value of “List (ls)”.

**Fail safe** :

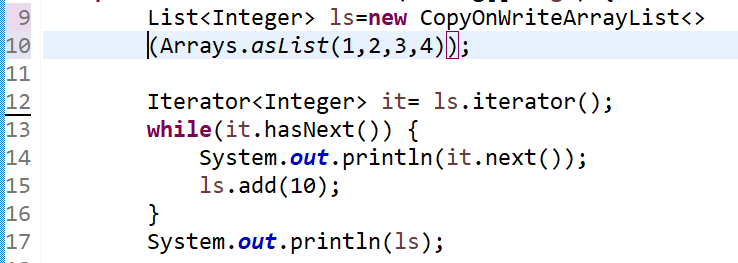
Those iterators create a clone of the actual Collection and iterate over it. If any modification happens after the iterator is created, the copy still remains untouched. Hence, these Iterators continue looping over the Collection even if it's modified

The Fail-Safe *Iterators* have a few disadvantages, though. One disadvantage is that the ***Iterator* isn't guaranteed to return updated data from the *Collection***, as it's working on the clone instead of the actual*Collection*.

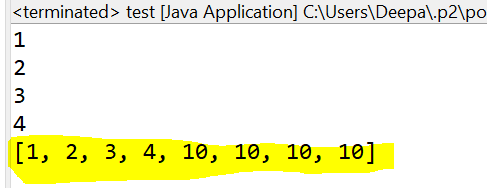
Another disadvantage is the overhead of creating a copy of the *Collection*, both regarding time and memory.

**Collection type** : ConcurrentHashMap***,***CopyOnWriteArrayList

**Code Example** :



**Output:**



Explain :

We are iterating the “CopyOnWriteArrayList” as well as adding a element 10 with each iteration but iteration not print 10 value which is adding after creating iterator object from the list;

Or we can say that after cloning the list.

**4. Q what is user of @RequestParam ?**

Ans :

**we can use @RequestParam to extract query parameters**

<http://localhost:4200/?id=1>**;**

**@GetMapping(“/”)**

**Public void f1(@RequestParam(“id”) Long id){**

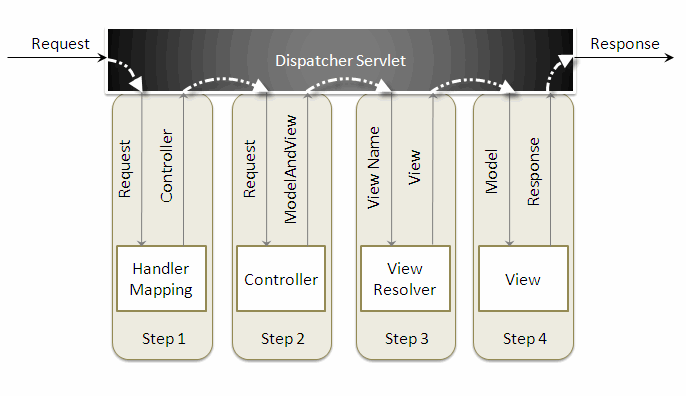
**}**

**5. Q what are the response Http response Code :**

**Ans :**

|  |  |
| --- | --- |
| 401 | Unauthorized client error |
| 403 | Forbidden client error status response code indicates that the server understood the request but refuses to authorize it |
| 500 | Internal Server Error server error response |
| 200 | Ok |
| 201 | Created |
| 503 | service unavailable |

**6. Spring MVC flow ?**

Ans . 

when a request is sent to the spring mvc framework the following sequence of events happen.

* the dispatcherservlet first receives the request.
* the dispatcherservlet consults the handlermapping and invokes the controller associated with the request.
* the controller processes the request by calling the appropriate service methods and returns a modelandview object to the dispatcherservlet . the modelandview object contains the model data and the view name.
* the dispatcherservlet sends the view name to a viewresolver to find the actual view to invoke.
* now, the dispatcherservlet will pass the model object to the view to render the result.
* the view , with the help of the model data, will render the result back to the user.

7. Q Deffernce between @Bean and @Autowired?

Ans.

* **@Bean :**
  + - @bean tells Spring 'here is an instance of this class, please keep hold of it and give it back to me when I ask'.
* **@Autowired** :
  + - @Autowired says 'please give me an instance of this class, for example, one that I created with an @Bean annotation earlier

8.Q what is defference between bean factory vs application context ?

Ans .

|  |  |
| --- | --- |
| Bean Factory | Application context |
| **BeanFactory loads beans on-demand** | **ApplicationContext loads all beans at startup** |
| BeanFactory does not register these interfaces automatically | **The ApplicationContext automatically registers BeanFactoryPostProcessor and BeanPostProcessor at startup** |
| **we should use BeanFactory only when memory consumption is critical** | More preferable to use because **ApplicationContext**comes with advanced features . also extends bean factory. |
|  |  |

**Q.9 The differences about BeanFactoryPostProcessor and BeanPostProcessor:**

Ans.:

1. . A bean implementing BeanFactoryPostProcessor is called when all bean definitions will have been loaded, but no beans will have been instantiated yet. This allows for overriding or adding properties even to eager-initializing beans. This will let you have access to all the beans that you have defined in XML or that are annotated (scanned via component-scan)
2. A bean implementing BeanPostProcessor operate on bean (or object) instances which means that when the Spring IoC container instantiates a bean instance then BeanPostProcessor interfaces do their work.
3. BeanFactoryPostProcessor implementations are "called" during startup of the Spring context after all bean definitions will have been loaded while BeanPostProcessor are "called" when the Spring IoC container instantiates a bean (i.e. during the startup for all the singleton and on demand for the proptotypes one)

**Q 10 . Dependancy IOC and DI ?**

Ans:

IOC :

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework.

IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. **If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes**

In the Spring framework, the interface ApplicationContext represents the IoC container. The Spring container is responsible for instantiating, configuring and assembling objects known as beans, as well as managing their life cycles.

The Spring framework provides several implementations of the *ApplicationContext* interface: *ClassPathXmlApplicationContext* and *FileSystemXmlApplicationContext*for standalone applications, and *WebApplicationContext* for web applications.

In order to assemble beans, the container uses configuration metadata, which can be in the form of XML configuration or annotations

**Advantages of IOC**

* decoupling the execution of a task from its implementation
* making it easier to switch between different implementations

**@Component // create object and maintain by IOC**

**Class A{**

**Public void f1(){**

**System.out.println();**

**}**

**}**

**@Component**

Class B{

@Autowired // portion of code

A aobj; // portion of code where ioc inject object of A class

public void f2(){

aobj.f1();

}

}

**Depandancy Injection :**

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.

3 types of DI

1. constructor
2. setter
3. field

Autowired annotation scan:

1.byName :

Match variable or property name with class name’s and autowired. Property name must be first letter small;

2. byType :

Match variable type

1. constructor :

Autowiring by constructor is similar to byType but it applies to constructor arguments. It will look for the class type of constructor arguments, and then do an autowire byType on all constructor arguments. If exactly one bean of the constructor argument type is not present in the container, a fatal error will be raised.

# Q 11 . ConcurrentHashMap in Java ?

# Ans:

* In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updated in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as **Segment locking or bucket locking**. Hence at a time, 16 update operations can be performed by threads.
* Inserting null objects is not possible in ConcurrentHashMap as a key or value .

**Q 12 .Why java not use pointer ?**

**Ans :**

**Manual memory management:**

you can use pointers to manually control and allocate blocks of memory . This is useful for some bigger applications like games, device drivers etc. but for general purpose Object Oriented programming it is simply not worth the effort. Java instead provides very good automatic Garbage Collection (GC) which takes care of memory management.

**Security:**

By not allowing pointers, Java effectively provides another level of **abstraction** to the developer. No pointer support make Java more secure because they point to memory location or used for **memory management** that loses the security as we use them directly.

**Q.13. JIT compiler ? (Just in time )**

Ans :

JVM use Interpreter and JIT compiler to convert the bytecode instructions into machine code.

For example

If we have an instruction which execute 1000 time. What JVM interpreter do convert this instruction 1000 time to machine code and run which impact performance .

On other hand JIT compiler compile more frequent instruction or method and store into the memory so next time . when every we need to execute that method we can directly call them without converting into machine code.

Ref : https://www.youtube.com/watch?v=UX2kf-XngE0&t=97s

**Q. 14 Why String is immutable in java ?**

Ans.

* String bool requires string to be immutable otherwise shared reference can be changed from anywhere.
* String are used in different area like file system, database connection ,having immutable string allows you to be secure and safe because no one can change reference once it created.

**Q. 15 What is marker interface and why we use that ?**

Ans :

Marker interface does not contain any data member and method signature.

Example . serializable

User for provide some meta data to jdk to do something special with thease.

**Q.16 Can we override and private or static method in java.**

Ans :

Private method :

You can not override private method private method only visible inside the class .

Static method :

You can not override static method if you create a method inside the child class it will simple hide the method of supper class this is also call method hiding.

Why not ? :

Static keywork refer to common to all objects if we override a static method then that method will be specific to that class . that’s why we are not can’t override static method , but we can access those method in child class.

For Example :

We have a parent class Mobile contain a static method “**setModel(String arg)”**  and child class Samsung which can use **setModel** method but can’t override that method . if Samsung class implement **setModel** method it will simple hide the method of **Moblie** class.

Ref : https://www.youtube.com/watch?v=PwiuAebCruY&t=605s

Q 17. Does Finally always excute in java ?

Ans . exception these two condition :

1. System.exit();
2. System crash

Ref : https://www.youtube.com/watch?v=PwiuAebCruY&t=605s

Q 18. What method does an Object class have ?

Ans.

Protected object clone() :

Return copy of the object

Public boolean Equals(object) :

Compare two object return result by default implementation of equals method use == operator and compare reference of objects

Protected void finalize():

Called by garbage collector on a object when garbage collector determines that there is not more reference to this object.

getClass():

this method return runtime class

getHashCode() :

this method return hash code this object

toString() :

return string implementation of object // default implementation return c lass@hashcode

Ref : https://www.youtube.com/watch?v=PwiuAebCruY&t=605s

Q 19. How can you make a class immutable?

Ans :

* Declare class as final so it can’t be extended.
* Make all field private and final
* Don’t provide setter
* Initialize all field via constructor
* If you are using any mutable field then in getter provide a clone copy of that field do not pass same reference. ( if you pass same reference this will break immutable pattern)



Ref : <https://www.youtube.com/watch?v=rptiI6qdCuI>

Ref : https://www.youtube.com/watch?v=vWklBcV2oIQ

Q.20 what is singleton class in java and how can we make a class singleton ?

Ans :

Singleton class is class whose one instance is created at any given time , in one JVM .

**public** **class** SingleTonClassExample {

**public** **static** SingleTonClassExample *obj*=**null**;

// create constructo private

**private** SingleTonClassExample() {

}

// create static getInstance() method

**public** **static** SingleTonClassExample getInstance() {

**if**(*obj*==**null**)

*obj*=**new** SingleTonClassExample();

**return** *obj*;

}

}

Q. Singletone class with double checking ?

Ans:

**class** A{

**public** **static** **volatile** A *obj*=**null**;

**private** A() {

}

**public** A getInstance() {

**if**(*obj*!=**null**) {

**synchronized** (A.**class**) {

**if**(*obj*!=**null**) {

*obj*=**new** A();

}

}

}

**return** *obj*;

}

}

Explain :

1. Create private constructor.
2. Create getinstance method which return object of “A class” .
3. Getinstance method create object using private constructor only if obj is null
4. First null check is required so if obj is not null then not need to create lock and directly return object.
5. Then we create a synchronized block so no two thread come at same time and create 2 object of this class
6. If I did not but 2nd null check then other waiting thread will also create object.
7. Create an object is 3 step of process
   1. Create empty object
   2. Assign resourse.
   3. Call construct.

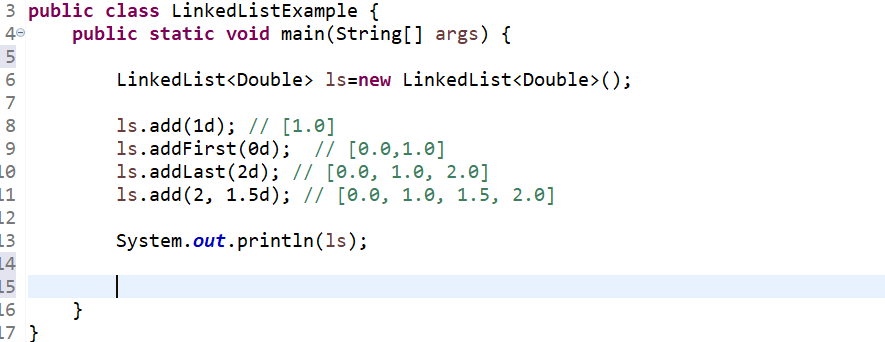
If any of step execute object will not null and other object can have empty object or assign resourced object.

To avoid is we use volatile keyword with object. so no thread keep this object in there cache and always read this from memory.

Q.21 what is difference between ArrayList vs LinkedList ?

Ans :

|  |  |
| --- | --- |
| ArrayList | Linked List |
| Internally it implements Array. | Interanally it implements Double Linked List |
| It is slow in manupuation | Fast in manupulation |
| It is fast in search | Slow in search |
| Only treat as List | You can use this as list,queue,stack |



Ref : https://www.youtube.com/watch?v=Iq9tUK-6J78

**Q. Arrays vs ArrayList ?**

Ans :

|  |  |
| --- | --- |
| Arrays | ArrayList |
| Arrays size is fixed and need to provide while created | ArrayList dynamic no need to provide while creating |
| Array support primitive and object , does not support generics. | ArrayList support object and generic , does not support primitive data type. |
| In Array you assign values to index | Where in ArrayList to add value at the end of list |
| To retive size of Array we use “.length” property | Where in ArrayList we need to call “.size()” method |

Q22. what is lazy initialization in hibernate ?

Ans. It will postpone initialization of an object as long as possible ?

For Example :

Address object in employee class and we have one to many mapping

On that with lazy initialization type lazy. when you try to fatch data of an employee address of that employee will not fatch until you try to access address object of that class.

Advantage : initial loading is fast

Disadvantage : unnecessarily hit on DB.

Q23. what is first level and second level cache in hibernate ?

Ans.

|  |  |
| --- | --- |
| First Level cache | Second level cache |
| By default fist level cache is enabled. | You need to programmatical configer second level cache . |
| First level chache associate with single transaction | Multiple transcation |
| You can not disable first level cache | You can disable this cache. |

Q24. How you implement 2nd level cache ? (my own knowledge )

Ans.

1. Ehcache dependancey
2. Enable caching (application properties)
3. Tell which caching framework we use
4. What to cache
5. @cacheable on entity class

Q25 . HashMap vs HashTable ?

|  |  |
| --- | --- |
| HashMap | HashTable |
|  |  |
|  |  |
|  |  |
| It uses equals() method of the Object class to compare keys. The equals() method of Map class overrides it. | It uses the **compareTo()** method to compare keys. |
| HashMap class contains only basic functions like **get(), put(), KeySet()**, etc. . | TreeMap class is rich in functionality, because it contains functions like: **tailMap(), firstKey(), lastKey(), pollFirstEntry(), pollLastEntry()**. |

Q26 . What is garbage collection in java and what is advantage of automatic garbage collection ?

Ans .

* Garbage collection is an automatic process of looking at heap memory. Identify which object are in use or which not, and delete unused object.
* An in use object , or a referenced object, mean that some part of program still refering that object.
* An unused object , which is no longer referenced by any part of program.
* Advantage of automatic garbage collection it remove the burden of manually allocating and deallocating the memory , so that we can foucs on business logic only.

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q27 . where does object store in memory ?

Ans :

Whenever an object is created, it always stored in heap and reference of the object store in stack memory.

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q.28. which part of memeory involved in garbage collection? stack or heap ?

Ans :

Garbage collection work on **Heap memory** where all newly created object store. Garbage collector delete unused object .

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

**Q29 . who manage garbage collector.**

Ans :

The JVM controls the garbage collector:

* JVM only decides when to run the garbage collector.
* JVM run the garbage collector when it realize it running low memory.
* You can request to garbage collector to happen within the java program but there is no guarantee

Jvm take care of your request or not.

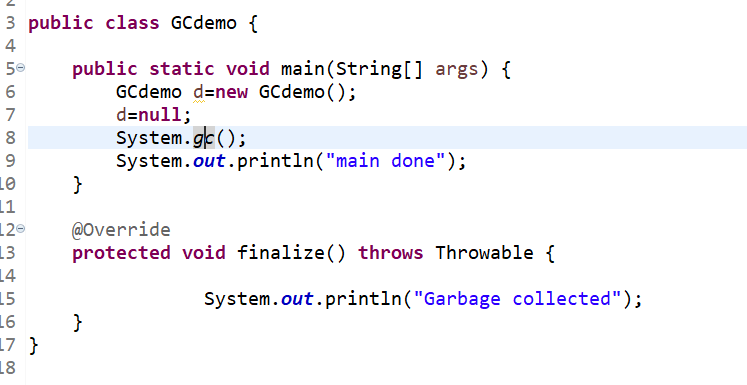
Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q.30 How can you request garbage collector ?

Ans : there are 2 ways to request garbage collector

System.gc()

Runtime.getRuntime().gc()



Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

**Q.31 When an object is eligible for garbage collection ?**

Ans :

When object has no reference from any part of program.

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

**Q.32 .How you can make object ready for garbage collection ?**

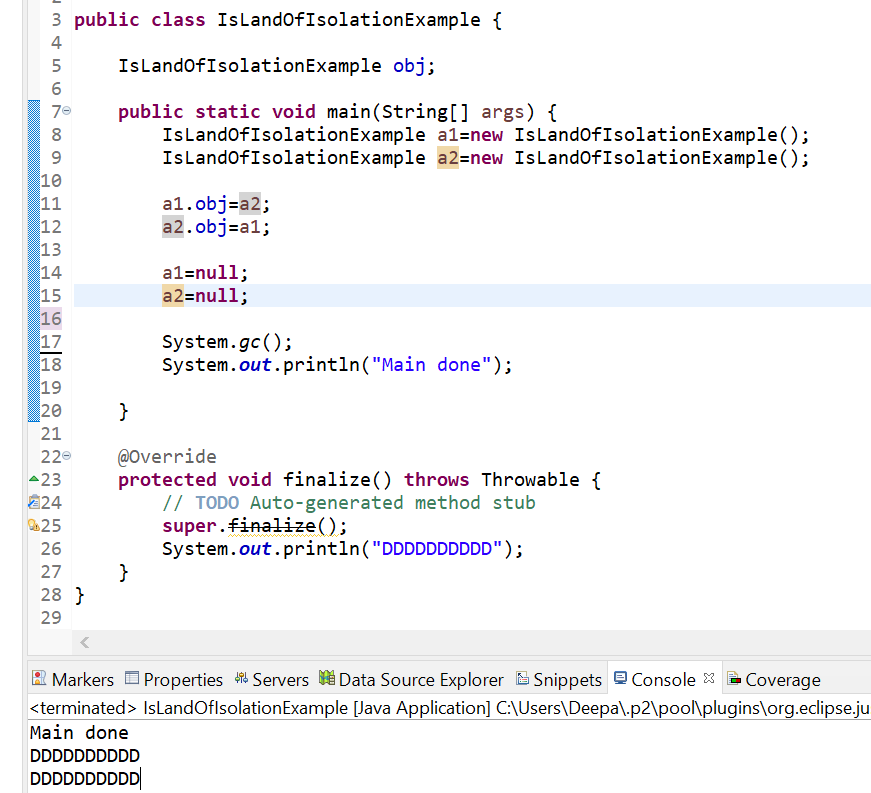
**Ans**

* **Point reference of object to null**
* **Point reference to another object**
* **Island of isolation**

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

**Q 33 .What is Island of isolation ? (GC)**

**Ans.**



Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q 34 . what is the perpouse of overriding finalize() method ?

Ans :

Finalize method will just call before and object goes for garbage collected. If you want to perfume sum sort of resource cleaning you can perform that in finalize method .

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q35 . How many time finalize() method called ?

Ans : Only once per object .

Q36 . what is responsibility of garbage collector ? ( prepare answere by own )

Ref : https://www.youtube.com/watch?v=EeRFOVMf7rE

Q 37 . Is garbage collector is background or foreground thread?

Ans.

* Garbage collector is background thread / Deam thread.
* Run behind the application
* It started by jvm
* And only stop when all foreground thread stopped.

Q38 . what is Serialization and deserialization ? why we need this ?

Ans :

Serialization :

Converting an object into bytestream is know as serialization. With the help of serialization we can store an object state in a file , DB or can transfer over the network.

For Example :

We have a student class with 2 properties “**Name**” and “**age**” . we create an object of this class with name=”**Deepak**” and age=**”25”** this is state of object. Once

We restart the program we loss the state or data . to prevent this we can store this object in file by converting this object into bytestream via serialization.

Deserialization:

Converting bytestream to an object is know as deserialization. With the help of deserialization we can retrive state of an object from file and DB and also over network.

Continue previous example :

when ever we want to student class object with name=**”deepak”** and age=”**25**” we need to read it byteStream to file or DB and need to deserialize to student class object .

Q. what are relationship in class (inheritance ,association ,IS-A,Has-A) ?

Ans:

Inheritance relations (Is – A):

When child class extends parent class this is knows as inheritance or IS-A relationship.

Example:

Class Employee extends from Men.

Employee **IS-A** Men.

**Note: tightly coupled**

Association relations (Has – A):

When class A create instance of another class within his scope. This relation is known as “**Has - A**”

Example :

Class Employee{

Address address; // employe has – A address

}

**Association has 2 type :**

1. **Aggregation (weak relation**) :

When two class have there own existence and can work independently.

Example :

Suppose CAR has a musical player. Car and musical player having Aggregation relation. Car can function without musical play. Musical player can function without car.

1. **Composition(strong relation):**

When two class are strongly connected with each other . and can’t function independently.

Example : CAR and Engines.

**Q. What is volatile Keyword does in java ?**

Ans.

As we know we can’t use synchronized keyword with variable that will raise compile time error. But we can use “Volatile” with key variable. When we use volatile keyword

With variable that multiple thread does not store this variable in there local cache as they do with normal variable. They need to read volatile variable from main memory or shared cache.

With this we get most recent updated value.

Volatile keyword does not maintain any lock.

Example :

Suppose you are have 2 unit of car manufacturing both unit work independently . and you have a task to manufacture 20 cars and both unit start working to complete task .

If both unit maintain localCounter for tracking manufactured cars then chances are high both unit generate 20 cars each and total car will be 40.

To avoid this problem we need a shared synchronized variable to track number of car manufactured by both unit.

But there is also a problem as we know volatile keyword does not lock variable so

Still multiple thread can read same value .

Suppose both unit come to check the counter and found counter is 19 and they start manufactering 1 car more then total number of cars 21.

To resolve this problem use synchronized block and atomic counter.

Atomic

Q. String vs StringBuffer vs StringBuilder ?

Ans .

|  |  |  |
| --- | --- | --- |
| String | String builder | String buffer |
| When you create an String object is created with literal that object created Inside string constant pool, only if not present in constant pool . Else you get reference of existing one.  Similarly when an object created with new key word it goes to heap memory . | store object in heap | |
| String are immutable | Stringbuilder is mutable | |
| When every I modify a string it will create a new object and store in memory due to this memory consumption is increase | In StringBuilder and StringBuffer modification done on same object itself. | |
| Operation or modification is thread safe. Because String is immutable. | Not synchronized and no thread safe | Synchronized and thread safe |
| Use String when you know  Value of string is not going to modify . | Use When single thread application and value of string need to modify more frequently | User Stringbuffer when working with multithread application and value of string need to modify more frequently |
| Slow as compare to string Buffer | Fast as compare to stringbuffer | Slow as compare to stringBuilder |

Q Shallow vs Deep Copy in Java

Ans :

Shallow copy :

Shallow copy provide you a copy of main object .not for the object inside the main object . if object inside the main object are mutable then changing state of object inner object will also impact the main object.

example :

I have a class person with 2 data members one is age which is “int” type and one is “Name” which is a mutable class having 2 data member “firstname” and “lastName” .

If, I use shallow copy of person object if I change age of new copy of object that will not impact the main object. But if I change name will impact the main object.



Deep copy :

Deep copy will not only create copy of main object but will copy inner object also. So that if you change any data member of copied object will not impact the main object.

For example:

Considering above example in deep copy even if you change name object that will not impact the main object.



Q Hashmap vs ConcurrenthashMap ?

|  |  |
| --- | --- |
| HashMap | Concurrent Hash Map |
| Non synchronized and no thread safe | Synchronized and thread safe |
| If you are iterating hash map and another thread maid change on hashmap then it will throw **ConcurrentModificationException** | It is will return a copy of current value in hashmap, even if concurrent hashmap change it will not throw error |
| As it is not synchronized and multiple thread can read and write same time without any lock. So it is faster then concurrenthashmap. | As it is synchronized and some time thread need to wait before read and write data particular segment. due to this it slower as comparative to hashmap. |

**Q comparable vs comparator ?**

**Ans :** Comparable :

We know that **Collections.sort()** sort method will help us to sort element in collection. But when we use custom class except Integer, String , Double etc.

**Collection.sort()** method won’t able to understand how to arrange object and throw an error.

to avoid this error we need to implement comparable interface and need to implement “ **int** **compareTo(Class obj)”** method .

Example :

We have an Employee class having variable id,salary,name **collection.sort** method don’t know whether it sort by **id** or **salary** or **name**.

So provide our custom logic to compare two object using comparable interface.

Comparator interface :

When did not implement comparable interface to custom class or we want to another logic to compare tow object. then we can use comparator interface . we use implement this interface as annonmus class.

Example :

Considering same example but now we want to sort the employee on base of salary in highest to lowest.

Code example comparable vs comparator



Q . What is polymorphism ?

Ans:

Ability of an object to take many forms. Any object which pass “IS-A” test is polymorphic and by default all object are polymorphic because every object is

Child of Object class .

For Example:

Suppose we have 2 class Friend and Employee these both class are child

Class of Person. Person can behave differently on the basis of different environment he present . (like with friend and with colleagues).

Q. What are the type of polymorphism?

Ans: there are two type of polymorphism:

1. Static polymorphism (overloading):

Static polymorphism does not required inheritance. when we have same method signature but argument of method are different (numbers, type). This is known as static polymorphism.

On the basis of parameter compiler decide which method need to call. this is done at compile time that’s why it is known as static.

You can not implement overloading on base of different return type.

2.Dynamic (runtime ) Polymorphism (Overriding) :

Dynamic polymorphism required inheritance. When parent class and child class having same signature

method . then compiler will decide which method need to call at runtime on basis of what class instance is refered by parent referenced variable.

Person{

Behave()

}

Friend extends Person{

Behave()

}

Employe extends Person{

Behave()

}

Runtime :

Person obj=null;

If(with friend){

Obj=new Friend();

}

If(with collegue){

Obj=new Employe()

}

Obj.behave();

Q overloading vs overriding ?

Ans

|  |  |
| --- | --- |
| Overloading | Overriding |
| Does not required inheritance to implement overloading | Required inheritance to implement overriding |
| Method argument must be different | Must be same |
| Which method need to call decide at compile time | Which method need to call decide at run time |
| Return type cannot be different | Can different but if parent class method return type “Object” then child class method return “String” ,”Integer” etc. |

Q . type promotion ? (overloading)

Ans :

Overloading :

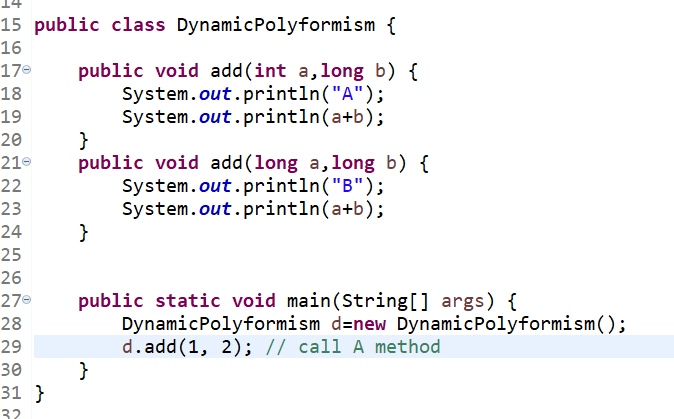
suppose we create a method “add(int a,long b)” and we call this method as “add(1,2)”

what happened in background autotype casting happen.

large data type easily store small data type. So “2” can easily store in b .

Following code example also shown :

One more point to note if you override an method “add(int a,int b)” then this method will call instad of “add(int a,long b)” because we know int take less byte to store “2” as compare to long.



Q. What is difference between dynamic polymorphism vs overriding ?

Ans :

Dynamic polymorphism is achieved overriding. Basic difference between between dynamic polymorphism vs overriding is How you creating reference variable .

Parent ojb=new child() // polymorphism

Child obj=new child() // overriding .

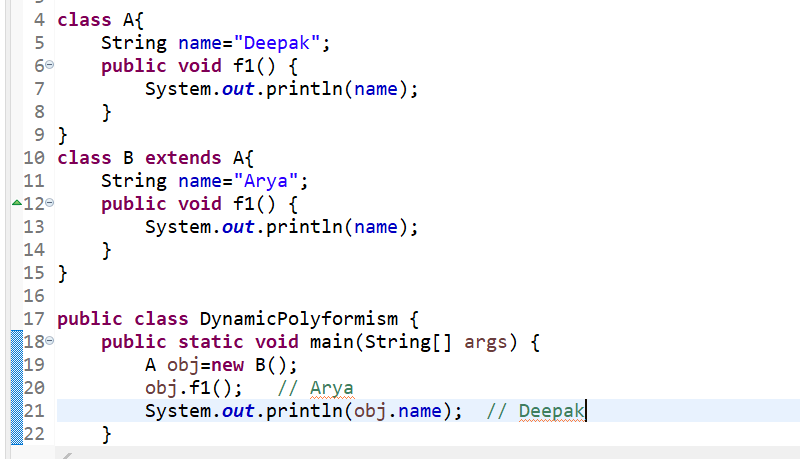
Calling method with parent reference variable point to child instance is know as Dynamic polymorphism.

Calling overriding method using child class instance is know as overriding.

Q. Can you override data member ?

Ans.

No you can not override data member.



Q . Why do we need Collections in Java?

Ans :

Arrays are not dynamic. Once an array of a particular size is declared, the size cannot be modified. To add a new element to the array, a new array has to be created with bigger size and all the elements from the old array copied to new array.

Collections are used in situations where data is dynamic. Collections allow adding an element, deleting an element and host of other operations. There are a number of Collections in Java allowing to choose the right Collection for the right context.

Q. Collection Interface hierarchy ?

Ans.

interface Collection<E> extends Iterable<E> {}

interface Set extends Collection { }

interface List extends Collection { }

interface Queue extends Collection { }

interface Map { }



Q . Equals method ?

Ans.

Default implementation of equals method compare two object on the basic of there memory location with equals operator ( == ) . even if two object are logical same but if they store in different memory location then equals method return false.

For example :

We have an Employee class with 2 variable like id,name; and we create two instance of Employee class with “new” keyword. Then both object store in different memory location . even if both having same data like id=1, name=Deepak . logical both object are same.

When we want to compare logical or on basic of value they contain we need to override “equals(Object obj)” method.

Note :

1. Compare two object on basis of memory location where the store are know as “Shallow compare”
2. Compare two object on the basic of values what the contain is known as “Deep compare”

Q. Contract between hashcode and Equals method ?

Ans:

1. If two object are equals according to equals method then hashcode must be same.
2. It is not necessary that if you have same hash code for 2 objects means that 2 object are equals . this is collision.
3. Hashcode method should return same integer throughout the java application unless values are changed.

Q. Why do we use default method ?

Ans .

Earlier version of java we can not define body of method inside interface and when ever we introduce new method in interface all the classes which implement this interface will start showing compile time error even if they have nothing to do with new method.

This increase head ache for developer to implement dummy implement on those classes. To remove this headache java 1.8 introduce default method . with the help of default feature of java 8 we can define body of method inside the interface.

All the class which has nothing to do with new method introduce in interface even they can also use default implmention of that method. Or also can override if they want.



Q. Working of class loader ?

Ans .

Step 1.

We know that JVM interpret code line by line , when a class is incounter first It whether this class is already loaded or not if already loaded then it use that class, else it request to “**class loader sub-system**” to load class.

Step 2 .

when class loader receive request from jvm it pass that to “**application class**”

application class does not load the class it pass this request to “**Extention class loader**”

“Extention class loader” also does not load the class it pass it to “**bootstrap class loader**”

**Note** : process of trasfering request from application -> extention -> bootstrap is kwon as “**delegation algorithm**”

Step3 :

“**bootstrap**” check class if it available or not, on the path “jdk/jre/lib/\*.jar” . if class is available then it load that class into memory. If class is not present at this path then control is moved to “**Extention class loader**”.

“**Extention class loader**” check class if it available or not, on the path “jdk/jre/lib/ext/\*.jar” . if class is available then it load that class into memory. If class is not present at this path then control is moved to “**Application class loader**”.

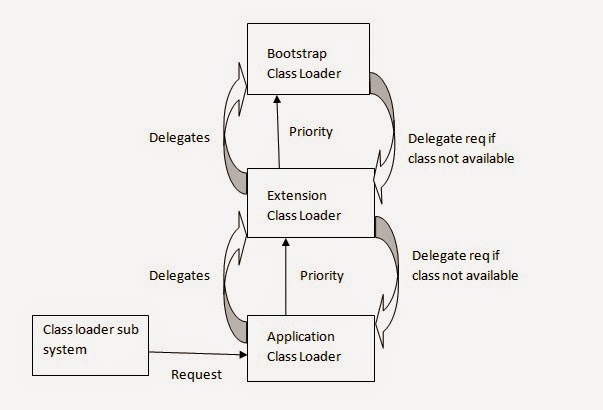
“**Application class**” check class if it available or not, on the path “**Application class path**” . if class is available then it load that class into memory. If class is not present at this path then it throw exception “**class not found**”.

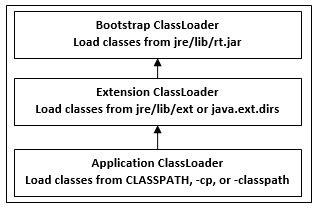
**Note** : process of trasfering request from bootstrap -> extention -> application is kwon as “**hierarchy algorithm**”

Example :

When jvm encounter “String.class” we if String class is not present memory then. Then it request to “**class loader subsystem**” to then request goes to “**application class loader**” then “**Extention class loader**” then “**bootstrap class loader**”

We know that “**String.class**” is present in “bootstrap classloader” path. So “Boostrap classloader” load “String.class” file and load into the memory.





Q. which class loader has higest priority ?

Ans :

Boot strap class loader has higest priority .

Q. Why hashset does not provide get method ?

Ans.

Hashset does not maintain any order of insertion (index) that’s why it does not provide “**get**” method .

Q. Hash Set internal working ?

Ans :

HashSet internally use hashmap. Firstly we discuss 4 constructor of hashset .

1. HashSet():

As we now hashset use hashmap internally this constructor initialize that hashmap with “new HashMap<>”

1. HashSet(int capacity) :

We know that hashmap has initial capacity is 16 by default . but with this constructor we can change the initial capacity of internal hashmap.

1. HashSet(int capacity, float loadFactor):

Default capacity is **16** and loadFactor is **0.75** with this consustructor we can change both .

1. HashSet(Collection<? extends E> c) :

If we want to add already create list or collection into hashset we can use this constructor.

**private** **static** **final** Object ***PRESENT*** = **new** Object();

**Add operation in hashset:**

Boolean add(Object key){

Return map.put(key,PRESENT)==null;

}

This will return true only once because when we firsttime add new element to hashmap it return null and next time if you try to put new element it will return that object.

**Remove operation** :

Boolean remove(Object key){

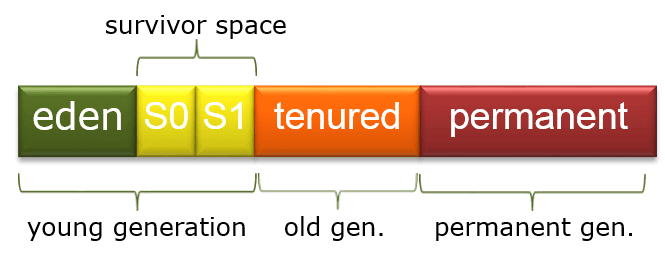
Return map.remove(key)==PRESENT;

}

Remove operation remove element from map and compare to “PRESENT” if both are equals then return true. If element is not present in list then it return null and return false.

Q . working of Garbage collector ?

Ans :



Q. How to break and prevent single ton class design pattern

Ans:

<https://www.youtube.com/watch?v=rSZIRsB0YlQ&t=5s>

reflection, readResolver(), clonable, throw exception from private constructo and clone method.

Q. difference between multitasking and threading.

Ans :

<https://www.youtube.com/watch?v=bHU4gDsDHLA>

Q. what is Thread ?

Ans.

<https://www.youtube.com/watch?v=bHU4gDsDHLA>

Q . Abstraction vs Encapsulation ?

Ans .

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| Abstraction is the process or method of gaining the information | While encapsulation is the process or method to contain the information |
| In abstraction, problems are solved at the design or interface level | While in encapsulation, problems are solved at the implementation level. |
| 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. |
| In abstraction, implementation complexities are hidden using abstract classes and interfaces. | Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public. |

Q. SerialVersionUid ?

Ans. https://www.youtube.com/watch?v=aHSzX16hGxM