CORE java

**Java transient keyword:**

The modifier **transient** in java can be applied to field members of a class to turn off serialization on these field members. Every field marked as **transient** will not be serialized. You use the **transient** keyword to indicate to the java virtual machine that the **transient** variable is not part of the persistent state of an object.

Let’s write an very basic example to understand what exactly above analogy means. I will create an Employee class and will define 3 attributes i.e. firstName, lastName and confidentialInfo. We do not want to store/save “**confidentialInfo**” for some purpose so we will mark the field as “**transient**“.

class Employee implements Serializable

{

   private String           firstName;

   private String           lastName;

   private transient String confidentialInfo;

   //Setters and Getters

}

o/p : will null

**What is the use of volatile keyword in java**

Use of volatile keyword in Java – The volatile keyword is used to declare a variable volatile, so, the variable can be accessed directly from the memory not from intermediary cache. For example,

If you declare a variable volatile, then it will be directly accessed from the memory where it resides and not from the intermediary any kind of cache.

**Example:** Let’s say there is an int variable that is shared by two threads. One is updating the value and other is reading the value. If the int variable is not volatile then there may be a chance that other threads that read the variable, don’t get updated value. If it is volatile, then for sure second thread will read the value directly from the memory and that will be updated one.

## 3. Java sleep() and wait() –

**sleep()** is a method which is used to pause the process for few seconds or the time we want to. But in case of **wait()** method, thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases the lock or monitor while waiting. wait() is used for inter-thread communication while sleep() is used to introduce pause on execution, generally.

# Java synchronized keyword

**Java synchronized keyword** marks a block or method a critical section. A critical section is where one and only one thread is executing at a time, and the thread holds the lock for the synchronized section.

**synchronized** keyword helps in writing [concurrent](https://howtodoinjava.com/java-concurrency-tutorial/) parts of the applications, to protect shared resources within this block.

The synchronized keyword can be use with –

* a code block
* a method

When a thread wants to execute synchronized statements inside the synchronized block, it MUST acquire the lock on lockObject‘s monitor. At a time, only one thread can acquire the monitor of a lock object. So all other threads must wait till this thread, currently acquired the lock, finish it’s execution.

In this way, synchronized keyword guarantees that only one thread will be executing the synchronized block statements at a time, and thus prevent multiple threads from corrupting the shared data inside the block.

# Loose coupling and tight coupling in java –

Tight coupling will be implemented using concrete implementation of classes and loose coupling will be achieved by using class and interface.

**loose coupling and tight coupling definition**  
Loose coupling means reducing dependencies of a class that use different class directly. Tight coupling means classes and objects are dependent on one another.

Let’s understand loosely coupled code and tightly coupled code by example and their advantages and disadvantages with this example only.  
Let’s consider a very simple example scenario**. There is a Manager and he wants to manage work of his workers. There are two workers, smart worker and lazy worker.**

Below are the classes for Manager, Smart worker and Lazy worker and complete program for this scenario. Code is self-understandable. Manager class have created objects of smart and lazy workers classes and have call them in ManageWork() method.

**TIGHTLY COUPLED**

/\*

\* Tight Coupling Example in java using concrete classes

class Manager {

SmartWorker sw;

LazyWorker lw;

public Manager(SmartWorker sw, LazyWorker lw) {

this.sw = sw;

this.lw = lw;

public void ManageWork() {

sw.work();

lw.work();

}

}

class SmartWorker {

public void work() {

System.out.println("smart worker working");

}

}

class LazyWorker {

public void work() {

System.out.println("Lazy worker working");

}

}

public class Test {

public static void main(String[] args) {

SmartWorker sw = new SmartWorker();

LazyWorker lw = new LazyWorker();

Manager mn = new Manager(sw, lw);

mn.ManageWork();

}

}

**Output:**  
smart worker working  
Lazy worker working

**So for sounds good. But let’s say requirement changes and we want to add one more worker that is Extraordinary Worker.**

class ExtraordinaryWorker {

public void work() {

System.out.println("ExtraOrdinary worker working");

}

}

Loosely coupled code reduces the cost of maintenance and efforts. Below is the same example implemented as a loosely coupled code using interface

**Notice that we are not going to touch the Manager class when we want to extend with Extra ordinary worker class.**

Just we will write new extra ordinary worker class and implement the interface. And in client program, we will call him.

We will create an interface IWorker and every worker will implement it.This program follows open-close principle and dependency Inversion.

**Loose Coupling Example**

/\*

\* Loose Coupling in java example

\*/

class Manager {

IWorker worker;

public Manager(IWorker worker) {

this.worker=worker;

}

public void ManageWork() {

this.worker.work();

}

}

interface IWorker{

void work();

}

class SmartWorker implements IWorker{

public void work() {

System.out.println("smart worker working");

}

}

class LazyWorker implements IWorker {

public void work() {

System.out.println("Lazy worker working");

}

}

class ExtraordinaryWorker implements IWorker{

public void work() {

System.out.println("ExtraOrdinary worker working");

}

}

public class Test {

public static void main(String[] args) {

SmartWorker sw = new SmartWorker();

Manager mn = new Manager(sw);

mn.ManageWork();

LazyWorker lw = new LazyWorker();

Manager mn2 = new Manager(lw);

mn2.ManageWork();

ExtraordinaryWorker ew = new ExtraordinaryWorker();

Manager mn3 = new Manager(ew);

mn3.ManageWork();

}

}

**Output:**

smart worker working  
Lazy worker working  
ExtraOrdinary worker working

**Conclusion:**

* Short definition of loose coupling and tight coupling in java is that loose coupling means reducing dependencies of a class that use different class directly. Tight coupling means classes and objects are dependent on one another.
* Disadvantage of tightly coupled code is that it took effort, testing and time that is reduced by writing loosely coupled coupled code.
* **tightly coupled code takes maintenance time and huge efforts.**

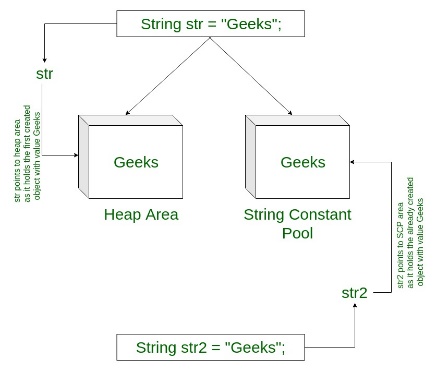
**NO**JAVA is not a pure object oriented programming language because :-

“ ***For any language to be pure OO everything should exist as objects but in JAVA we have eight primitive data types such as int, byte ,char etc which are not objects . “***

But on the other hand you can make a pure object oriented JAVA program because along with primitives JAVA also supports wrapper classes where we can use Integer ,Float etc instead of int,char etc which are objects so a JAVA program which contains wrapper instead of primitive datatypes will be a pure OO program but on a whole JAVA is not a pure object oriented programming language because of the primitives.

* String(This is what string constant pool and heap)

Whenever a String Object is created, two objects will be created- one in the Heap Area and one in the String constant pool and the String object reference always points to heap area object.



e.g String str=”Greek”: In this scenario , first JVM will check this object is present or not in constant pool , if it is not present then create it new object.

String str1=new String(“Greek”)

In this scenario, JVM creates two object one is in heap and second is in constant pool but if Greek is already present in above scenario so it will not create new object in constant pool but it will create new object into heap..

**Encapsulation vs Data Abstraction**

1. [Encapsulation](http://contribute.geeksforgeeks.org/encapsulation-in-java/) is data hiding(information hiding) while Abstraction is detail hiding(implementation hiding).
2. While encapsulation groups together data and methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of implementation.

What is shallow and deep copy

Cloning is a process of creating a replica or copy of [java](https://www.edureka.co/blog/java-tutorial/) object, clone method Java.lang.Object is used to create copy or replica of an object. java objects which implement Cloneable interface are eligible for using the clone method.

## **Creating Copy of Java Object**

We can create a replica or copy of java object by

1. Creating a copy of object in a different memory location. This is called a Deep copy.

2. Creating a new reference that points to the same memory location. This is also called a Shallow copy.

## **Shallow Copy**

The default implementation of the clone method creates a shallow copy of the source object, it means a new instance of type Object is created, it copies all the fields to a new instance

This object will have an exact copy of all the fields of source object including the primitive type and object references. If the source object contains any references to other objects in field then in the new instance will have only references to those objects, a copy of those objects is not created. This means if we make changes in shallow copy then changes will get reflected in the source object.

e.g

A a=new A();

a.i=10;

A a1=a.clone();

a1.i=20;

So here a1.i will get reflected in a.i as well bz both are pointing to same memory location

## **Deep Copy**

The deep copy of an object will have an exact copy of all the fields of source object like a shallow copy, but unlike sallow copy if the source object has any reference to object as fields, then a replica of the object is created by calling clone method. This means that both source and destination objects are independent of each other. Any change made in the cloned object will not impact the source object.

\***How HashMap work internally**

It works on the hashing functionality, what does it means, it has 3 stuffs

1)hascode(): Hashcode of key that memory reference of object in integer form

2)hash(): this function returns the hash code .

3)Bucket : IndexFor () -IndexFor(hashcode, size-1)

where n is number of buckets or the size of array. In our example, I will consider n as default size that is 16

when the collision gets occurred then what happen

e.g: map.put(“vishal”,”123”): index is 6

again, I am inserting map.put(“shalvi”,”456”);now index is 6 but index 6 has already an entry of vishal key so collision gets occurred then how next entry will get stored internally.

1. In that case, check via hashCode() and equals() method that if both the keys are same.
2. If keys are same, replace the value with current value.
3. Otherwise connect this node object to the previous node object via linked list and both are stored at index 6.  
   Now HashMap becomes

When we fetch the map element using get(): it uses both equal() and hashcode(): we already 6 index has two entries present

Vaibhav

Vishal

6

While fetching the get(Vaibhav) first it will find the bucket location then will find the hascode of ”Vaibhav” yes, hashcode is same then it will check equal() for key if the key is same then it will return the value of key.

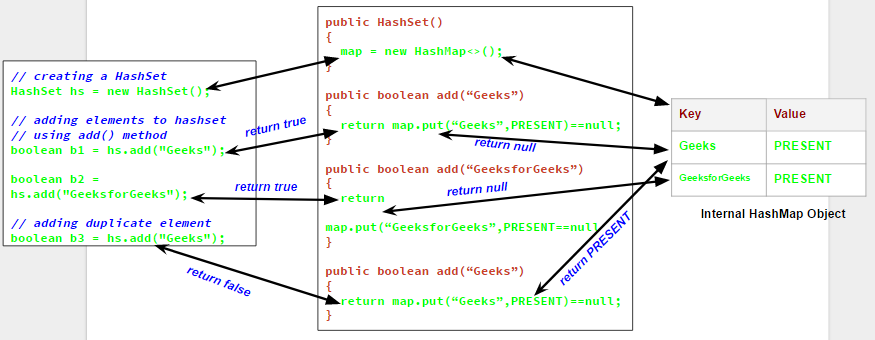
We want to retrieve the shalvi key’s value

1. Bucket location of this is 6 but 6 has 2 entries then how will it return the value.
2. Calculate hash code of Key {“vaibhav”}. It will be generated as 118.
3. Calculate index by using index method it will be 6.
4. Go to index 6 of array and compare first element’s key with given key. If both are equals then return the value, otherwise check for next element if it exists.
5. In our case it is not found as first element and next of node object is not null.
6. If next of node is null then return null.
7. If next of node is not null traverse to the second element and repeat the process 3 until key is not found or next is not null.

If there are other objects in the bucket with same hash code, then “key” equals method comes into play.

* If equals() return true and it’s a put operation, then object value is overridden.
* If equals() return false and it’s a put operation, then new entry is added to the bucket.
* If equals() return true and it’s a get operation, then object value is returned.
* If equals() return false and it’s a get operation, then null is returned.
* When binary tree gets used in java 8:
* Capacity\*load factor i.e 16\*0.75
* It depends on the load factor. If it's 0.75, it means after 75% of the storage, it will create 16 more segments. But by default it will always create 16 segments for hash map.
* **For example** product of capacity and load factor as 16 \* 0.75 = 12. This represents that after storing the 12th key – value pair into the HashMap , its capacity becomes 32.
* ***In java 8 when binary tree gets comes into picture when 4 index has 8 nodes then it creates binary tree.***
* ***Complexity of hash Map java 7 is O(n) (it takes 16 steps)and java 8 is O(log n)i.e. it takes very less time to perform the operation***

**\*How HashSet works internally:**



As we see in above diagram, Geeks is not added by hashset and it returns as false. what happened internally when the duplicate elements are passed to the HashSet? The add(e) method in HashSet returns false when the element exists in the HashSet, otherwise it returns true.

If map.put(key,value) returns null, then map.put(e, PRESENT)==null will return true and element is added to the HashSet.

If map.put(key,value) returns the old value of the key, then map.put(e, PRESENT)==nullwill return false and element wont be added to the HashSet.

\***What is difference between HashMap and concurrentHashmap**

* HashMap:

1)It is non-synchronized. It allows null key. Even if it is synchronized , you can make it synchronized using Collections.synchronizedMap().

2)It is not a thread safe(i.e running multiple threads and one thread trying to modify)

* ConcurrentHashMap:

1)It is synchronized and it does not allow null key.

2)It is thread safe i.e one thread can perform modification while the other thread is running.

* In HashMap, if one thread is iterating the object and the other thread wants to modify the objects, we will get a **ConcurrentModificationException** runtime exception. But, in ConcurrentHashMap, one thread can perform modification while the other thread is running.

What is marker interface

1)Cloneable: It is an empty interface, it does not have any method but if your cloning any object.you should override the clone method and implement cloneable interface . if your not implementing cloneable interface but trying to clone method then it will throw **ClassNotSupportedException** ().it is in java.lang package

2)Serializable: convert object into byte called serialization and byte to object is called de-serialization. It is java.io package

Questions:

1) What is method hiding?

2) What is use of static import?

3) What is difference between hash map and concurrenthashmap.

4) How can we make immutable class?

5) What happen, if string is mutable?

6) What is up casting and auto boxing?

7) What is singleton?

8) How hash map works internally.

9)What is synchronization in java?

10)What is string pool and heap.

e.g String s1=new String(“test”);

it creates two object , 1-is in string pool and second is in heap.

11)what is use of string intern();

: it is used when u want to reduce the duplications of object.

e.g-String s2=s1.intern(): it will return string pool string.-i.e test

**Interning of String in Java:**

String Interning is a method of storing only one copy of each distinct String Value, which must be immutable.  
By applying String.intern() on a couple of strings will ensure that all strings having the same contents share the same memory. For example, if a name ‘Amy’ appears 100 times, by interning you ensure only one ‘Amy’ is actually allocated memory. Intern() method will be true, if equals() method returns true

e.g

String str1="Teju";

String str2="Teju";

System.***out***.println("The intern method:"+(str1.intern()==str2.intern()));

* Java 8:
* **What is Functional Interface?**
* If a Java interface contains one and only one abstract method then it is termed as functional interface. This only one method specifies the intended purpose of the interface. For example, the Runnable interface from package java.lang; is a functional interface because it constitutes only one method i.e. run().

From Java 8 onwards, [lambda expressions](https://www.geeksforgeeks.org/lambda-expressions-java-8/) can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. ***Runnable***, ***ActionListener***, ***Comparable*** are some of the examples of functional interfaces.  
Before Java 8, we had to create anonymous inner class objects or implement these interfaces.

1. A functional interface has only one abstract method but it can have multiple default methods.
2. @FunctionalInterface annotation is used to ensure an interface can’t have more than one abstract method. The use of this annotation is optional.

## **Why use Lambda Expression**

What is lambda expression?

Lambda expression is anonymous function which have set of parameters and a lambda (->) and a function body .You can call it function without name.

1. To provide the implementation of Functional interface.
2. Less coding.
3. @FunctionalInterface  //It is optional
4. **interface** Drawable{
5. **public** **void** draw();
6. }
7. **public** **class** LambdaExpressionExample2 {
8. **public** **static** **void** main(String[] args) {
9. **int** width=10;
10. rawable d2=()->{
11. System.out.println("Drawing "+width);
12. };
13. d2.draw();
14. }

What are the new features of java 8?

**1)Functional interfaces**: before java 8 also there was functional interface

So if we want to override the interface abstract method , we were overriding it using anonymous class . so lambda is used to provide the implementation of functional interface . e.g Runnable interface which has run method which is abstract. In java 8 has predict , Consumer interface

**2) Default method:**

By adding default method in interface, you can provide default implementation of it without affecting implementing classes as it includes implementation of that method and any implementing class which needs that method can override it.

The default methods were introduced to provide backward compatibility so that existing interfaces can use the lambda expressions without implementing the methods in the implementation class. Default methods are also known as defender methods or virtual extension methods.

->Using default method, you can override the default method in derived class as well.

->One of the major reason for introducing default methods in interfaces is to enhance the Collections API in Java 8 to support lambda expressions.

**3)Java Interface Static Method:**

Java interface static method is similar to default method except that we can’t override them in the implementation classes.

### **forEach() Method In Iterable Interface**

Java 8 has introduced a “forEach” method in the interface java.lang.Iterable that can iterate over the elements in the collection. “forEach” is a default method defined in the Iterable interface.

### **Optional:**

Optional< String > fullName = Optional.ofNullable( null );

System.out.println( "Full Name is set? " + fullName.isPresent() );

he **isPresent()**method returns **true** if this instance of **Optional** has non-null value and **false** otherwise.

#### **Question: How is Spliterator a different Iterator?**

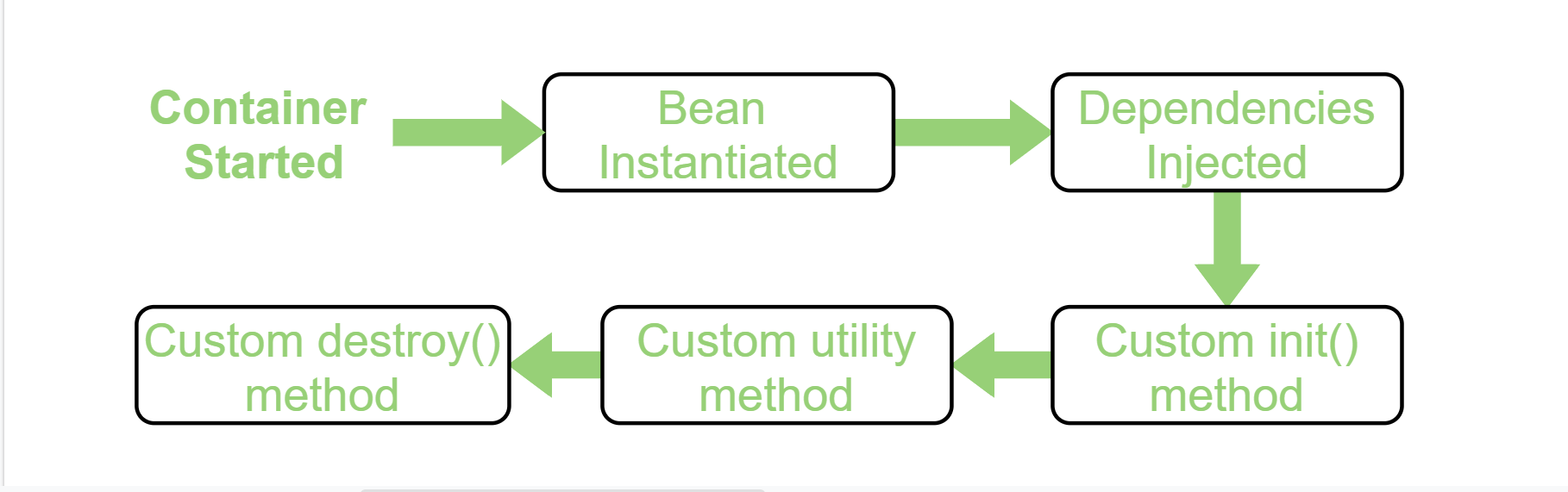
**Answer:**The following are the differences between Spliterator and Iterator.

|  |  |
| --- | --- |
| Spliterator | Iterator |
| It is introduced along with Java SE 8 | It was introduced long ago with Java 1.2 |
| It is defined as a Spliterator iterator. | It is defined as a non-spilterator Iterator. |
| It can be used in Stream API. | It can be used in Collection API. |
| It helps to iterate streams in parallel and sequential order. | It helps in iterate collections only in sequential order. |
| The examples include tryAdvance() | The examples include, next(), hasNext(). |

**How to create custom annotation?**

What is solid principles?

**Bean life cycle in Java Spring:**



**Using Annotation:** To provide the facility to the created bean to invoke custom **init()** method on the startup of a spring container and to invoke the custom **destroy()** method on closing the container, we need annotate **init()** method by **@PostConstruct** annotation and **destroy()** method by **@PreDestroy** annotation.

**Note:** To invoke the **destroy()** method we have to call the **close()** method of ConfigurableApplicationContext.

Therefore, the following steps are followed:

* Firstly, we need to create a bean HelloWorld.java in this case and annotate the custom init() method with @PostConstruct and destroy() method with @PreDestroy.

|  |
| --- |
| // Java program to create a bean  // in the spring framework  package beans;    import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;    // HelloWorld class  public class HelloWorld {        // Annotate this method to execute it      // automatically as the bean is      // instantiated      @PostConstruct      public void init() throws Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm the "              + "init() method");      }        // Annotate this method to execute it      // when Spring container is closed      @PreDestroy      public void destroy() throws Exception      {          System.out.println(              "Conatiner has been closed "              + "and I'm the destroy() method");      }  } |

**Web services:**

Added security-constrains into web.xml where we defined access privileges for different role with resource URL

How ibatis is configured in LUMIS:

1)Sql-map-config-dao.xml: Added ibatis query xml

<sqlMapConfig>

<sqlMap resource=”cm/java/ibatis/ResinstMiscComDetails.xml”/>

</sqlMapConfig>

2)spring-ibatis-dao.xml

It has mentioned datasource i.e jdbc.Lumis4 is used and initializing all the the above sql-map-config-dao.xml

e.g

<bean id=”dataSource” class=”org.framwork.jndi.jndiObjectFcatoryBean”

P:jndiName=”jdbc/lumis4” p:proxyInterface=”javax.sql.DataSource”/>

<bean id=”anbDataAccessSqlMapClient” class=”org.framwork.orm.ibatis.sqlMapClientFcatoryBean”

P:configLocation=”/spring/ sql-map-config-dao.xml”

P:dataSource-ref=”dataSource”/>

<bean id=’reinstMiscImpl’ class=”com.genworth.net.ibatis.db.dao.ReinstMiscImpl”/>

Then all the daoimpl need to config into this file

1) What is sprint, stories and how do u follow agile methodologies.

2) Is @autowired annotation instantiate the bean?

It does not instantiate the bean. It just activates the register bean

**3) What is diff between requestMapping and getMapping();**

ANS: This annotation can be used both at the class and at the method level. And Annotation for mapping HTTP GET requests onto specific handler methods.

4) What is spring actuator?

5) What are the advantage and disadvantage of micro service architecture.

6)What is the purpose of controller annotations.

7) Difference between <context:annotation-config> and <context:component-scan>

ANS: