**Real Interview Question**

**Core Java**

**1. Diff b/w jdk and jre**

* **The JRE is the Java Runtime Environment. It is a package of everything necessary to run a compiled Java program, including the Java Virtual Machine (JVM), the Java Class Library, the java command, and other infrastructure. However, it cannot be used to create new programs.**
* **The JDK is the Java Development Kit, the full-featured SDK for Java. It has everything the JRE has, but also the compiler (javac) and tools (like javadoc and jdb). It is capable of creating and compiling programs.**

1. **JDK is for development purpose whereas JRE is for running the java programs.**
2. **JDK and JRE both contains JVM so that we can run our java program.**
3. **JVM is the heart of java programming language and provides platform independence.**
4. **When we execute a Java program, JVM is responsible for converting the byte code to the machine-specific code.**

**2. What is wrapper classes?**

* **A Wrapper class is a class whose object wraps or contains primitive data types.**
* **When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.**

**Need of Wrapper Classes**

1. **The classes in java.util package handles only objects and hence wrapper classes help in this case also.**
2. **Data structures in the Collection framework, such as**[**ArrayList**](https://www.geeksforgeeks.org/arraylist-in-java/)**and**[**Vector**](https://www.geeksforgeeks.org/vector-vs-arraylist-java/)**, store only objects (reference types) and not primitive types.**
3. **An object is needed to support synchronization in multithreading.**

****

* 1. **what is immutable class ? how to create custom immutable class**
* **once we create the object the content will not change.**
* **If we are trying to the change the content with those change new object will be created.**
* **If no change in the content, existing object will be reused.**
* **Eg: string, wrapper classes**

**Create own immutable class**

1. **Declare the class as final.**
2. **Make all its fields final and private.**
3. **For all mutable fields, make sure that the class creates a copy and only returns the copy to the calling code.**
4. **Create parameterized constructor to initialize the final variable.**
5. **Do not provide any setter methods.**

**4.Difference between final and immutable class**

* **Final is related to class, immutability is related to object**
* **Final class also will allow to change the object content**
* **Hence even we declare string buffer reference as final, it will allow to change content.**
* **If reference variable is final we can’t reassign with new object.**

**Eg:**

**Final StringBuffer sb= new StringBuffer(“test”);**

**Sb.append(“final”)// s.o.p(sb); -- test final**

**Sb = new StringBuffer(“new”)// CE ,, we cant reassign final references**

**5.What is string class, methods available in string class**

* **Strings are used to store a sequence of characters in Java, they are treated as objects.**
* **The String class of the java.lang package represents a String.**
* **String is a final and immutable class**
* **It cannot be inherited, and once created, we can not alter the object.**
* **You can create a String either by using the new keyword (like any other object) or, by assigning value to the literal (like any other primitive datatype).**
* **Strings are stored on the heap area in a separate memory location known as String Constant pool. String constant pol: It is a separate block of memory where all the String variables are held. String str=”hello”;**
* **And whenever we try to create another String as String str2 =”hello”; JVM verifies whether any String object with the same value exists in the String constant pool, if so, instead of creating a new object JVM assigns the reference of existing object to the new variable.**
* **Methods:**

[**https://www.w3schools.com/java/java\_ref\_string.asp**](https://www.w3schools.com/java/java_ref_string.asp)

**6. String Object creation Heap Area vs SCP**

* **You can create a String either by using the new keyword (like any other object) or, by assigning value to the literal (like any other primitive datatype).**
* **Strings are stored on the heap area in a separate memory location known as String Constant pool. String constant pool: It is a separate block of memory where all the String variables are held. String str=”hello”;**
* **And whenever we try to create another String as String str2 =”hello”; JVM verifies whether any String object with the same value exists in the String constant pool, if so, instead of creating a new object JVM assigns the reference of existing object to the new variable.**
* **And when we store String as String s = new String (“hello); a new object with the given value is created irrespective of the contents of the String constant pool.**
* **Also when we create string object using new keyword, one object will be created in String constant pool as well for future reference.Eg:**

**String s1 = new String(”hello”);**

**String s2 = “hello”;**

**s.o.p(s1==s2); // false**

* **bcz string s1 created in Heap area and SCP for future purpose,**
* **S2 already available in SCP area. Hence S1 and S2 created in different area and different object so it not equal.**

**String s3 = “hello”;**

**s.o.p(s2==s3); // true**

* **S3 is a string literal which is already available in SCP, hence s3 and s2 pointing to same objects in SCP.**

**String S5= “he”+”llo”;**

**s.o.p(s3==s5); // true**

* **Here hello and world are constant which is concatenated (+) at run time by JVM**
* **JVM will look for hello in SCP area which is already created for s3. so s5 point to that object**
* **If both are constant literals then this operation is performed by compile time**

**String S6= “he”;**

**String s7=s6+”llo”;**

**s.o.p(s3==s7); // false**

* **Here He is a new string literal will be created in SCP.**
* **If any one literal is normal variable this operation performed at run time and object created in heap area.**
* **Hence hello object created in heap area and pointing to s7.**

**Final String S8= “he”;**

**String s9=s8+”llo”;**

**s.o.p(s5==s9); // true**

* **Now “he” already available in SCP area.**
* **All final variable will be replaced by value at compile time only. means for s9 all constant is replaced at compile time like “he”+”llo”**
* **The + operation performed by JVM at run time and it is created in SCP which is equivalent to s5.**

**7.Different b/w String & String Buffer**

**1. Mutablity vs Immutablity**

**●     String objects are immutable - non changeable- once u created object , we can’t change existing objects**

**●     String buffer objects are mutable - changeable - once we created object , we can change the existing objects at any time.**

**Eg 1: string s= new String(“durga”);**

**s.concat(“software”);**

**s.o.p(s); // durga**

* **Here string is used. so , concat not change existing ‘s’ value. Also it is not assigned to any variable. So o/p is durga only**
* **If content is fixed and won't change frequently then go for String**

**Eg 2: StringBuffer sb= new StringBuffer(“durga”);**

**sb.append(“software”);**

**s.o.p(sb); //durga software**

* **Here software appended with existing sb object. which means it is changeable**
* **Hence it is called mutability**
* **If content is not fixed and keep on changing, then go for string buffer or string builder**

**2. equals() vs == operator**

* **In object class equals() method is used to compare the reference/address**
* **If both object reference pointing to the same then only it returns true.**
* **In child class we can overridden equals method for content comparison.**
* **In String class equals method is overridden to compare content but in string buffer class it is not overridden.**
* **Hence in string class equals() used to content comparison and string buffer class equals() used to reference comparison**
* **In both string and string buffer class == is used for reference comparison**

**8. equals() method in StringBuffer()/ StringBuilder()**

* **In object class equals() method is used to compare the reference/address**
* **If both object reference pointing to the same then only it returns true.**
* **In child class we can overridden equals method for content comparison. Eg: String Class**
* **In StringBuffer equals method not overridden. Hence parent class (Object class) equals method will call which is address comparison.**
* **The same happened in string builder**
* **We can compare the content by converting the buffer object into string using toString().**

**9.string buffer, string builder**

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| **1)** | **StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.** | **StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.** |
| **2)** | **StringBuffer is *less efficient* than StringBuilder.** | **StringBuilder is *more efficient* than StringBuffer.** |
| **3)** | **StringBuffer was introduced in Java 1.0** | **StringBuilder was introduced in Java 1.5** |

**10.Difference between final, finally, finalize**

## **Final**

* **It is a keyword.**
* **It is used to apply restrictions on classes, methods and variables.**
* **Final class can’t be inherited.**
* **Final method can’t be overridden.**
* **It is needed to initialize the final variable when it is being declared or in constructor.**
* **Its value, once declared, can’t be changed or re-initialized.**

## **Finally**

* **It is a block.**
* **It is used to place important code in this block.**
* **It gets executed irrespective of whether the exception is handled or not.**

## **Finalize**

* **It is a method.**
* **It is used to perform clean up processing right before the object is collected by garbage collector.**

**11. Explain autoboxing and unboxing.**

* **The automatic conversion of primitive data types into its equivalent Wrapper type is known as auto boxing**
* **reverse operation is known as unboxing.**
* **So java programmer doesn't need to write the conversion code.**

**Advantage:**

**No need of conversion between primitives and Wrappers manually so less coding is required.**

**12.What is serialization VS Deserialization**

**Note : https://www.geeksforgeeks.org/serialization-in-java/**

* **The process of converting object from java supported form into network or file supported form is known as serialization**
* **Serialization is a mechanism of converting the state of an object into a byte stream.**
* **Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object**
* **The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.**
* **To make a Java object serializable we implement the java.io.Serializable interface.**
* **The ObjectOutputStream class contains writeObject() method for serializing an Object.**
* **The ObjectInputStream class contains readObject() method for deserializing an object.**

**13. what is serial version id**

* **SerialVersionUID is a unique identifier for each class, JVM uses it to compare the versions of the class ensuring that the same class was used during Serialization is loaded during Deserialization.**
* **Defining a serialVersionUID field in a serializable class is not mandatory. ... If there is no serialVersionUID field defined explicitly then serialization runtime will calculate default value for that class.**
* **The value can vary based on compiler implementation. Hence it is advisable to define serialVersionUID**
* **private static final long serialVersionUID = 4L;**

**14. where we can use the serializable Interface how to implement**

* **while storing data into Database, when we create entity pojo class we have to implements Serializable Interface to send data in network (Encryption)**
* **When we write the content into file or read the file content we will use the serialization**
* **Static, transient data member can’t be serialized , if we want to restrict few instance data member not to serialized we can make it transient.**
* **Eg:**

**@Entity**

**@Table(name = "GMPLUS", schema = "dbtest")**

**@IdClass(GMPlusId.class)**

**public class GMPlusEntity implements Serializable**

**{**

**// The Constant serialVersionUID**

**private static final long serialVersionUID = -3371371230940094635L;**

**// The gm ID pri4y column**

**@Id**

**@Column(name = "gm\_id")**

**private int gmID;**

**}**

**15.Different between volatile and transient key word**

* **A volatile keyword is used in a multithreading environment where two threads reading and writing the same variable simultaneously. The volatile keyword flushes the changes directly to the main memory instead of the CPU cache.**
* **On the other hand, the transient keyword is used during serialization. Fields that are marked as transient can not be part of the serialization and deserialization. We don't want to save the value of any variable then we use transient keyword with that variable.**

| **Sr. No.** | **Key** | **Volatile** | **Transient** |
| --- | --- | --- | --- |
| **1** | **Basic** | **Volatile keyword is used to flush changes directly to the main memory** | **The transient keyword is used to exclude variable during serialization** |
| **2.** | **Default value** | **Volatile are not initialized with a default value.** | **During deserialization, transient  variables are initialized with a default value** |
| **3** | **Static** | **Volatile can be used with a static variable.** | **Transient can not be used with the static keyword** |
| **4** | **Final** | **Volatile can be used with the final keyword** | **Transient can not be used with the final keyword** |

**16.What is garbage collection ? What happens if you say System.gc()?**

**Ref:** [**https://www.geeksforgeeks.org/garbage-collection-java/**](https://www.geeksforgeeks.org/garbage-collection-java/)

* **The purpose of garbage collection is to delete unused objects**
* **Sun people provided one feature which always run-in background (Daemon thread) and destroy useless objects.**
* **Bcz of this chance of failing java program with memory problems is very very low. This feature is Garbage Collector.**
* **System. gc() method runs the garbage collector. Calling this suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse.**

**16.What is marker interface?**

* *A marker interface is an*[*interface*](https://www.baeldung.com/java-interfaces)*that****has no methods or constants inside it****. It provides****run-time type information about objects****, so the compiler and JVM have****additional information about the object****.*
* *A marker interface is also called a tagging interface.*
* *Java has many built-in marker interfaces, such as Serializable, Cloneable, and Remote.*
* *Let's take the example of the*[Cloneable*interface*](https://www.baeldung.com/java-deep-copy)*. If we try to clone an object that doesn't implement this interface, the JVM throws a*CloneNotSupportedException*. Hence, the*Cloneable***marker interface is an indicator to the JVM****that we can invoke the*Object.clone()*method.*
* *In the same way, when invoking the*ObjectOutputStream.writeObject()*method,****the JVM checks if the object implements the*Serializable *marker interface****. When it's not the case, a*NotSerializableException*is thrown. Therefore, the object isn't serialized to the output stream.*

**17. static vs instance block**

|  |  |
| --- | --- |
| ***Static block*** | ***instance block*** |
| Known only as [**static**](http://www.javamadesoeasy.com/2015/05/static-keyword-in-java-variable-method.html) **initialization block** in java. | Also known as **non-static initialization block** in java. |
| **static blocks** executes **before instance blocks** in java. | **instance blocks** executes **after static blocks** in java**.** |
| Only static variables can be **accessed** inside **static block** | Static and non-static variables (instance variables) can be **accessed** inside **instance block.** |
| **static blocks** can be used for **initializing** static  **variables**  **or**  invoking any static **method** in java. | **instance blocks** can be used for initializing instance **variables**  ***or***  invoking any instance **method** in java. |
| **static blocks** executes **when class is loaded** in java. | **instance block** executes only **when instance of class is created**, not called when class is loaded in java. |
| **this** [**keyword**](http://www.javamadesoeasy.com/2015/05/keywords-in-java-language.html) cannot be used in **static blocks**. | **this keyword** can be used in **instance block**. |

**18. Can we define enum inside a method?**

**No enum we can use inside the class not inside method**

**19. explain about hashCode() and equals() method**

**1.** [**hashCode() method**](https://www.geeksforgeeks.org/equals-hashcode-methods-java/)

* **hashCode() method is provided by object class used to get the hash Code value of an object.**
* **hashCode() method of object class returns the memory reference of object in integer form.**
* **It is possible to provide your own implementation of hashCode().**
* **In HashMap, hashCode() is used to calculate the bucket index.**

**2.** [**equals() method**](https://www.geeksforgeeks.org/equals-hashcode-methods-java/)

* **equals method is used to check that 2 objects are equal or not.**
* **This method is provided by Object class. You can override this in your class to provide your own implementation.**
* **HashMap uses equals() to compare the key whether they are equal or not. If equals() method return true, they are equal otherwise not equal.**

**20. Contract between hashCode and equals**

* **If two Objects are equal according to equals(Object o) method then the hash code for both the object must be the same.(memory address in Integer value)**
* **if we have same hash code for 2 objects means , its not necessary that , those two object are equal. This is hash Collision**
* **Whenever it is invoked on the same Object more than once during an execution of a java application, the hash code method must consistently return the same integer**

**21. Why/when we override hashcode(),equals () method?**

* **The equals method is used to check whether two objects are same or not.**

**It needs to be overridden if we want to check the objects based on property.**

**For ex: employee is a class that has 3 data members: id, name and salary. But we want to check the equality of employee object on the basis of name. then, we need to override the equals , hashcode method**

1.When we create user defined class we need to override equals & hashcode method

2. If we are not override equals () then default object class implementation will be considered which will compare memory reference not content

3. In set and map it should not allow duplicate, it will  avoid duplicate by comparing the newly added value with existing value with equals method

4. We need to define the equality of user defined object in equals () , once we override equals method we should override hashcode properly as per hashcode & equals contract

5. If we didn't override hashcode , equals method return true for 2 obj , but hascode return different memory address for 2 object, hence different bucket index calculated and both object will be added to set or map even its equals method returns true,

Its violates the contract between equals and hashcode methods which is if 2 objects are equal according to equals method then the hashcode for both must be same

**22.Shallow Copy vs Deep Copy**

* **Object can be create by copying the existing object, the resultant object is object copy or simply copy of original object**
* There are several ways to copy an object, most commonly by a [copy constructor](https://www.geeksforgeeks.org/copy-constructor-in-java/) or [cloning](https://www.geeksforgeeks.org/clone-method-in-java-2/).
* Cloning means create a copy of object.
* Shallow copy, deep copy , lazy copy is related to cloning process
* **Cloning process is used to create copy of existing object. the resultant object is object copy or simply copy of original object.**
* **Similar to cloning we can use copy constructor to create a copy of object,**

**Ref: https://www.javatpoint.com/shallow-copy-vs-deep-copy-in-java**

**Shallow copy:**

* **In shallow copy only reference is copied**
* **When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.**

**Deep Copy:**

* **In deep copy new object is created and changes applied in original obj will not affect copied object**
* **When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.**

|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| **It is fast as no new memory is allocated.** | **It is slow as new memory is allocated.** |
| **Changes in one entity is reflected in other entity.** | **Changes in one entity are not reflected in changes in another identity.** |
| **The default version of the clone() method supports shallow copy.** | **In order to make the clone() method support the deep copy, one has to override the clone() method.** |
| **A shallow copy is less expensive.** | **Deep copy is highly expensive.** |
| **Cloned object and the original object are not disjoint.** | **Cloned object and the original object are disjoint.** |

**Lazy Copy**

**23. Which block will be executed first static, constructor, instance**

* A **static block** is a block of code with a static keyword. In general, these are used to initialize the static members of a class. JVM executes static blocks before the main method at the time loading a class.
* **Instance Initialization Blocks** are used to initialize instance variables. IIBs are executed before constructors. They run each time when object of the class is created.
* A **constructor** is similar to method and it is invoked at the time creating an object of the class, it is generally used to initialize the instance variables of a class. The constructors have same name as their class and, have no return type.
* Instance methods are the normal methods of a class (non static), you need to invoke them using an object of the class –
* Order of execution – Static block, Main method, Instance block, constructor, instance method, Static method
* **Static method comes before object creation then it execute before instance block.**

**Eg: package** com.wm\_practice.example;

Prgrm: OrderOfExecution

15. Shallow vs Deep compare

**24. how to clone on object:**

* **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.**
* **Default implementation of clone() method in Java provides "*shallow copy"* of the object because it creates a copy of Object by creating a new instance and then copying content by assignment, which means if your class contains a mutable field, then both original object and clone will refer to the same internal object.**
* **This can be dangerous because any change made on that mutable field will reflect in both the original and copy object. In order to avoid this, override the clone() method to provide the**[**deep copy of an object**](http://javarevisited.blogspot.sg/2014/03/how-to-clone-collection-in-java-deep-copy-vs-shallow.html)**.**

**25.different ways to create object in java**

**Ref:** [**https://www.geeksforgeeks.org/different-ways-create-objects-java/**](https://www.geeksforgeeks.org/different-ways-create-objects-java/)

1. **Using new keyword – A obj = new A();**
2. **Using new instance –**

**Class cls = Class.forName("GFG");**

**// Creating object of main class**

**// using instance method**

**GFG obj = (GFG)cls.newInstance();**

**3.Using clone() method -             GFG obj2 = (GFG)obj.clone();**

1. **Using deserialization –**

**Whenever we serialize and then deserialize an object, JVM creates a separate object. In deserialization, JVM doesn’t use any constructor to create the object. To deserialize an object we need to implement the Serializable interface in the class.**

1. **Using newInstance() method of Constructor class**

**26.What is Reflection?**

**Ref:** [**https://www.geeksforgeeks.org/reflection-in-java/**](https://www.geeksforgeeks.org/reflection-in-java/)

* **Reflection is used to examine or modify the behavior of methods, class, interfaces at runtime.**
* **The required classes for reflection are provided under java.lang.reflect package**

**27.Solid Principles:**

**Ref:** [**https://www.javatpoint.com/solid-principles-java**](https://www.javatpoint.com/solid-principles-java)

**28. this keyword**

* The this keyword **refers to the current object in a method or constructor**.
* The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

**OOPS**

**.**

1. **What is Object class**

* **The Object class is the parent class of all the classes in java by default.**
* **In other words, it is the super class of all java classes**
* **The Object class is beneficial if you want to refer any object whose type you don't know.**
* **Notice that parent class reference variable can refer the child class object, known as upcasting.**
* **Downcasting?**

1. **What are the methods available in Object class**
2. **toString() – return value of string object**
3. **equals(Object obj) – it is used to compare the reference/address**
4. **hashCode() – return the hashcode value(integer form of object reference )**
5. **clone() – it return clone copy of object**
6. **finalize() – it will perform clean up activities for the unused object before it is destroyed.**
7. **getClass()**
8. **wait()**
9. **notify()**
10. **notifyAll()**
11. **Different between object based and oops**

**4. OOPs concepts**

* **Data hiding – it is hiding of data, outside person can’t see the data directly, to achieve use private**
* **Abstraction - *Hiding internal details and showing functionality* is known as abstraction**
* **Encapsulation - Process of binding data members and corresponding behavior into a single unit is encapsulation. The whole idea behind encapsulation is to hide the implementation details from users**
* **Inheritance - Inheritance is a mechanism in which one class acquires the property of another class.**
* **Polymorphism - Polymorphism in Java is a concept by which we can perform a single task in multiple ways**
* **Has a relationship – one class has a reference of another class. Teachers has department reference**
* **Composition – objects are strongly associated. university has college , without university college does not exist.**
* **Aggregation -** If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship. objects are weekly associated. Without Library Student can exist.
* **Method signature - Method signature contains only method name followed by argument types. here returns type is not part of method signature**
* **Overloading - two methods are said to be overloaded, if and only if both methods having same name but different argument types**
* **Overriding - Whatever method parent has by default is available to the child through inheritance. If the child class is not satisfied with parent class implementation, then, the child is allowed to redefine that method based on its requirement. This process is called overriding**

**5.what is encapsulation**

* **Process of binding data members and corresponding behavior into a single unit is encapsulation.**
* **The whole idea behind encapsulation is to hide the implementation details from users.**
* **If a data member is private it means it can only be accessed within the same class. No outside class can access private data members (variables) of other classes.**
* **If any component follows data hiding and abstraction and its is encapsulated component**
* **Eg:java class**

**6.what is polymorphism**

* **Polymorphism in Java is a concept by which we can perform a single task in multiple ways**
* **There are two types of polymorphism in Java**
  + - 1. **compile-time polymorphism / static polymorphism /early binding**

**Eg : method overloading , method hiding**

* + - 1. **runtime polymorphism / dynamic polymorphism / late binding**

**Eg: method overriding.**

| **Sr. No.** | **Key** | **Compile-time polymorphism** | **Runtime polymorphism** |
| --- | --- | --- | --- |
| **1** | **Basic** | **Compile time polymorphism means binding is occurring at compile time** | **R un time polymorphism where at run time we came to know which method is going to invoke** |
| **2** | **Static/Dynamic Binding** | **It achieved through static binding** | **It can be achieved through dynamic binding** |
| **4.** | **Inheritance** | **Inheritance is not involved** | **Inheritance is involved** |
| **5** | **Example** | **Method overloading is  an example of compile time polymorphism** | **Method overriding is an example of runtime polymorphism** |

**7.What is overriding?**

* **Overriding allows to provide specific implementation of child class methods which is already provided by parent class.**
* **Parent class and child class method should be same name with same parameter, method signature should be same**
* **Child class method is overriding method. Parent class method is overridden method.**

**8.What is overloading ? Why should we go for over loading?**

* **If a**[**class**](https://www.javatpoint.com/object-and-class-in-java)**has multiple methods having same name but different in parameters, it is known as Method Overloading.**
* **Methods can have different return types**

**Advantage:**

* **The main advantage of this is cleanliness of code.**
* **Method overloading increases the readability of the program.**
* **Overloaded methods give programmers the flexibility to call a similar method for different types of data.**
* **Overloading is also used on constructors to create new objects given different amounts of data.**
* **You must define a return type for each overloaded method.**

**9.Difference between overloading and overriding**

**Overloading**

**1. method name must be same**

**2. argument type must be different, at least order**

**3. method signature(method name and argument) must be different**

**4.no restriction on return type**

**5. private, static, final method overloaded**

**6. no restriction for access modifiers**

**Overriding**

**1. Method name must be same**

**2. Argument types must be same including order**

**3. Method signature must be same**

**4. Covariant return types are allowed; child class return type may be parent class child type or return type should be same**

**5.Overriding not applicable for private, static (method hiding) , final**

**6.Same or high-level scope of modifiers allowed.eg: parent default, child – default, protected, public**

**10.Difference between interface and abstract class**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Abstract class** | **Interface** |
| **1** | **Abstract class supports abstract, final methods, static and non static methods** | **interface doesn’t support final methods, but it supports abstract, static, default methods** |
| **2.** | **It supports static, non static,final, non final variable** | **It support public static final variable only** |
| **3.** | **It supports no of constructors** | **It doesn’t support the constructor** |
| **4.** | **It supports private, protected, public** | **It is not support private and protected members. All members are public by default** |
| **5.** | **It is not support multiple inheritance** | **It supports multiple inheritance** |
|  |  |  |
|  |  |  |

**11.what is singleton class, how it is used in real-time, how to create singleton class.**

**Singleton Class**

* **A Singleton class in Java allows only one instance to be created and provides global access to all other classes through this single object or instance.**
* **The primary purpose of Single class is to restrict the limit of the number of object creation to only one. This often ensures that there is access control to resources, for example, socket or database connection.**
* **The memory space wastage does not occur with the use of singleton class because it restricts the instance creation. As the object creation will take place only once instead of creating it each time a new request is made.**
* **Singleton classes are used for logging, driver objects, caching and thread pool, database connections.**

**To create a singleton class using Lazy Initialization method, we need to follow the below steps:**

* **Declare the constructor of the class as private.**
* **Create a private static instance of this class but don’t initialize it.**
* **In the final step, create a factory method. This method will first check whether the instance member is null or not.**
* **If it is not null, then it will create an instance of the singleton class for you and return it; otherwise, it will not create any instance.**

**Code:**

**public class LazyInitialization**

**{**

**// private instance, so that it can be**

**// accessed by only by getInstance() method**

**private static LazyInitialization instance;**

**public String string;**

**private LazyInitialization ()**

**{**

**// private constructor**

**string = "Welcome to TechVidvan's Tutorial of Java";**

**}**

**//method to return instance of class**

**public static LazyInitialization getInstance()**

**{**

**if (instance == null)**

**{**

**// if instance is null, initialize**

**instance = new LazyInitialization ();**

**}**

**return instance;**

**}}**

**12. What is constructor? Types of constructor/ What is default and parameter constructor**

**Constructor:**

* **A constructor initializes an object when it is created.**
* **It has the same name as its class and is syntactically similar to a method.**
* **However, constructors have no explicit return type.**
* **Typically, you will use a constructor to give initial values to the instance variables defined by the class,**

**class Test {**

**Test() {**

**// constructor body**

**}m**

**}**

**In Java, constructors can be divided into 4 types:**

1. **No-Arg Constructor - If a constructor does not accept any parameters, it is known as a no-argument constructor**
2. **Parameterized Constructor - A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).**
3. **Copy Constructor- A copy constructor in a Java class is a constructor that creates an object using another object of the same Java class. That's helpful when we want to copy a complex object that has several fields, or when we want to make a deep copy of an existing object.**
4. **Default Constructor - If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor. if we create any parameterized / copy constructors JVM will not create default constructor.**

**Note: Deep copy/ cloning is the process of creating exactly the independent duplicate objects in the heap memory and manually assigning the values of the second object where values are supposed to be copied is called deep cloning.**

**13. what is covariant return type**

* Child class method return type need not to be the same as parent class method,
* If parent method return type is Object then subclass override method return type (String) can be child of Object.
* In simple words override method return type should be child of overridden method return type.
* If parent is string and child is Object then it throws error
* This co-variant is applicable for non-primitive only, not primitive data types
* Eg: parent double and child int is not applicable

**14.** Difference between static binding & dynamic binding ?

**14. Aggregation Vs Composition – video required**

**Exception Handling**

**1.What is exception handling**

* **An exception is an unwanted event that interrupt the normal flow of the program**
* **When an exception occurs, program gets abnormally terminated and system generated error messages throws By handling this exception we can provide meaningful error message which user can understand.**

**2.Type of exceptions**



**1.Checked Exception**

* + - **All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not.**
    - **If these exceptions are not handled/declared in the program, you will get compilation error.**
    - **For example, SQLException, IOException, ClassNotFoundException etc.**
  + **Partial Checked exception – Exception (some child classes are un checked), Throwable (Error is un checked exception)**
  + **Fully checked exception - IO Exception,**

**2.Unchecked Exception**

* **Runtime Exceptions are also known as Unchecked Exceptions.**
* **These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it’s the responsibility of the programmer to handle these exceptions**
* **For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc**

**3.Difference Checked and un checked exception**

|  |  |
| --- | --- |
| **Checked Exception** | **Unchecked Exception** |
| Checked exceptions occur at compile time. | Unchecked exceptions occur at runtime. |
| The compiler checks a checked exception. | The compiler does not check these types of exceptions. |
| These types of exceptions can be handled at the time of compilation. | These types of exceptions cannot be a catch or handle at the time of compilation, because they get generated by the mistakes in the program. |
| They are the sub-class of the exception class. | They are runtime exceptions and hence are not a part of the Exception class. |
| Here, the JVM needs the exception to catch and handle. | Here, the JVM does not require the exception to catch and handle. |
| Examples of Checked exceptions:   * File Not Found Exception * No Such Field Exception * Interrupted Exception * No Such Method Exception * Class Not Found Exception | Examples of Unchecked Exceptions:   * No Such Element Exception * Undeclared Throwable Exception * Empty Stack Exception * Arithmetic Exception * Null Pointer Exception * Array Index Out of Bounds Exception * Security Exception |

**4.Difference between ClassNotfound exception and No Class Def found error**

|  |  |
| --- | --- |
| **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 class path. | 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 class path is not updated with required JAR files. | It occurs when required class definition is missing at runtime. |

1. **Difference between throw and throws**

| **Sr. No.** | **Key** | **Throw** | **Throws** |
| --- | --- | --- | --- |
| **1** | **Definition** | **Throw is a keyword which is used to throw an exception explicitly in the program inside a function or inside a block of code.** | **Throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code.** |
| **2** | **Internal implementation** | **Internally throw is implemented as it is allowed to throw only single exception at a time i.e we cannot throw multiple exception with throw keyword.** | **On other hand we can declare multiple exceptions with throws keyword that could get thrown by the function where throws keyword is used.** |
| **3** | **Type of exception** | **With throw keyword we can propagate only unchecked exception i.e checked exception cannot be propagated using throw.** | **On other hand with throws keyword both checked and unchecked exceptions can be declared and for the propagation checked exception must use throws keyword followed by specific exception class name.** |
| **4** | **Syntax** | **Syntax wise throw keyword is followed by the instance variable.** | **On other hand syntax wise throws keyword is followed by exception class names.** |
|  |  |  |  |

**Pg ex:** [**https://www.tutorialspoint.com/difference-between-throw-and-throws-in-java**](https://www.tutorialspoint.com/difference-between-throw-and-throws-in-java)

**6. Custom exception handling? How to create custom exception handling class**

**Ans:**

* **Custom exception is used to throw the exception explicitly**
* **If we are creating our own Exception that is known as custom exception or user-defined exception.**
* **Java custom exceptions are used to customize the exception according to user need.**
* **By the help of custom exception, we can have your own exception and message.**
* **We can create custom exception class by Extending Exception class or Run time exception**
* **To create checked custom exception By extending Exception class**

**Eg: public class MASDataBaseException extends RuntimeException**

**{**

**}**

**7.What finally block will do in exception handling?**

* A **finally block** contains all the crucial statements that must be executed whether exception occurs or not.
* The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.
* A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.
* Finally block is optional, however if you place a finally block then it will always run after the execution of try block.
* In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
* An exception in the finally block, behaves exactly like any other exception.
* The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue.

**8. What happens if both try, finally block has return statement?**

* **Before executing return statement in try block , it will execute finally block**
* **Hence only finally block return statement executed not try block return statement.**

**9.What will happen if we replace exception by throwable class**

* **Throwable is the super class of error and exception**
* **If we extend / use throwable in catch block it will catch all exceptions and errors**
* **It is not required to throw or handle erros(Out of memory error, stack overflow error)**

**10. what is the Output we have return statement in all try catch finally and outside try catch finally ?**

* **It will throw CE for return statement outside try catch**
* **If we removed those return , if exception occur catch block return statement work**
* **If exception not works finally block return statement works**
* **Try block return statement never executed.**

**10.How to identify and resolve the memory leaks in java?**

**Multithreading**

1. **What is multi-threading**

* **The process of executing multiple threads simultaneously is known as multithreading.**
* **All threads of a process share the common memory.**
* **It is save cpu time**
* So, threads are light-weight processes within a process

1. **Syntax to create custom multithreading class**

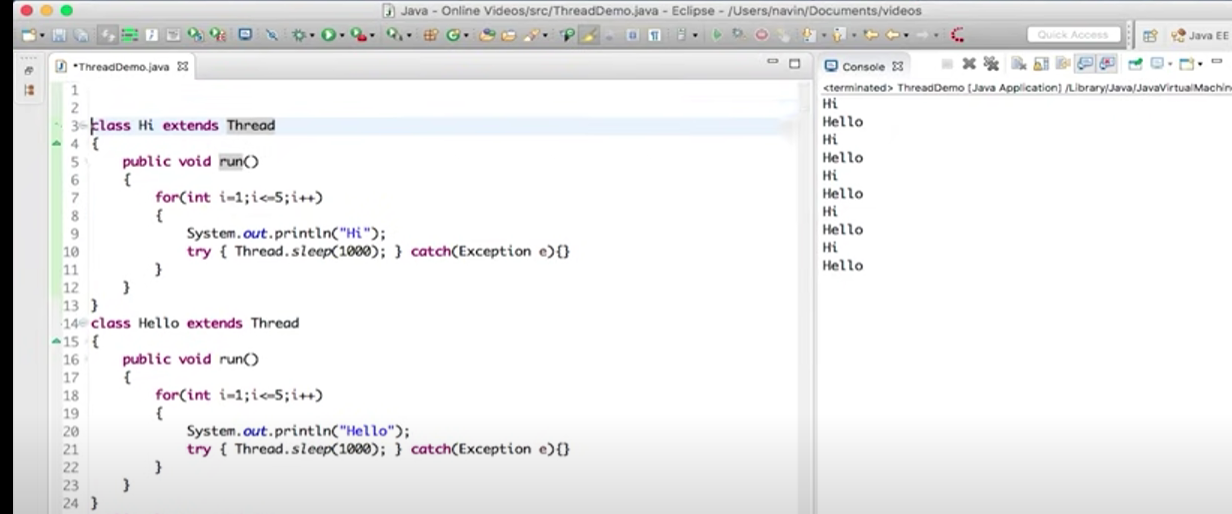
**Public class MyThreadClass extends Thread{**

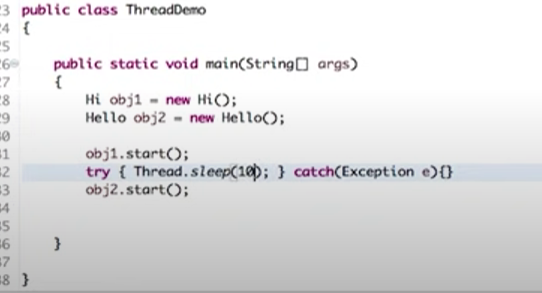
**Public void run(){**

**// start method internally invoke run method, so our implementation goes here**

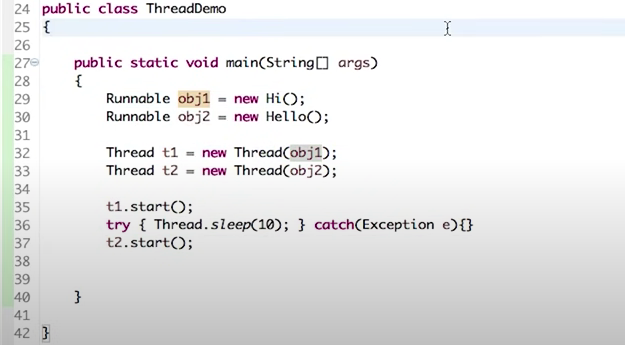
**}**

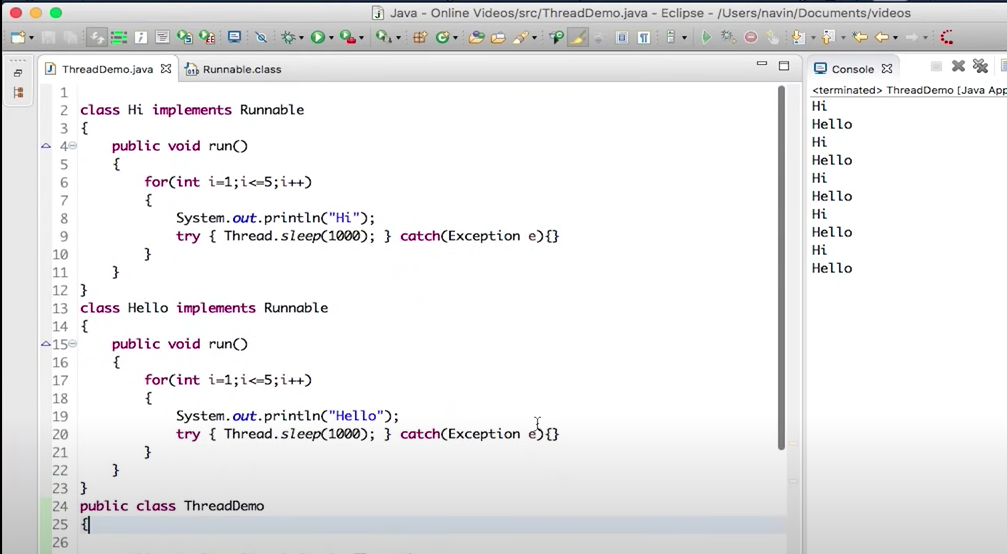
**}**

1. **Ways of creating thread object?**
   1. **By using thread class**



* 1. **By implements runnable interface**





1. **Thread lifecycle**
2. **How you will start the thread in multi threading**

**MyThread t = new MyThread();**

**t.start();**

**(or)**

**Runnable onj1= new Hi();**

**Thread t = new Thread(obj1);**

**t.start();**

**When a program calls the start() method, a new thread is created and then the run() method is executed. But if we directly call the run() method then no new thread will be created and run() method will be executed as a normal method call on the current calling thread itself and no multi-threading will take place.**

1. **Difference between start() , run()**

| **START()** | **RUN()** |
| --- | --- |
| **Creates a new thread and the run() method is executed on the newly created thread.** | **No new thread is created and the run() method is executed on the calling thread itself.** |
| **Can’t be invoked more than one time otherwise throws *java.lang.IllegalStateException*** | **Multiple invocation is possible** |
| **Defined in *java.lang.Thread* class.** | **Defined in *java.lang.Runnable* interface and must be overriden in the implementing class.** |

1. **What is daemon thread**
2. **what is synchronization types of synchronization**

**●     In multi threading program sometime multiple thread may access the same resource at the same time, finally it produce the erroneous output**

**●     So by using synchronization we can ensure that only one thread can access the resource at a time**

**●      It can be achieved by using synchronized keyword**

**Types:**

**1.synchronized method -** synchronized void print(){}- entire method will get locked for thread

**2.synchronized block –** synchronized(this){} – only block of code is locked so perfomance are improved than synchronized method

**3. static synchronized method – static** synchronized return type class name{} = **provides class level lock not instance level lock.**

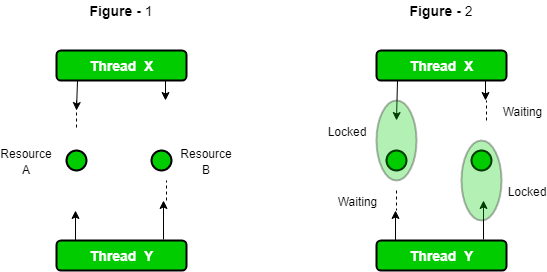
1. **Difference between synchronized method & synchronized block?**

* synchronized block reduce scope of lock, **but** synchronized method's scope of lock is whole method.
* synchronized block has better performance as only the critical section is locked **but** synchronized method has poor performance than block.
* Waiting time is high in synchronized method and low in synchronized block.
* synchronized block provide granular control over lock **but** synchronized method lock either on current object represented by this or class level lock.
* synchronized block can throw NullPointerException **but** synchronized method doesn't throw.
* **synchronized block:** synchronized(this){}
* **synchronized method:** public synchronized void fun(){}

1. **How to make one thread wait for other two among three threads?**
2. **What is Lock & sleep method**

1. **Difference between synchronization and serialization**
2. **What is deadlock , how to resolve it.**

**Deadlock describes a condition in which two or more threads are blocked (hung) forever because they are waiting for each other.**



**Collection**

* 1. **What is Collection, what is collection Framework**
* **collection represents group of individual objects as a single entity**
* **qCollection is used to overcome the limitation of Array (fixed size, homogeneous data, readymade method not available)**
* **collections are growable in nature, i.e based on our requirement we can increase/decrease the size**
* **collection can hold both homogeneous /heterogeneous elements**
* **every collection class is implemented based on some standard data structure. hence readymade method support is available for every requirement. being a programmer we have to use this method and we are not responsible to provide implementation.**
* **Collection Framework that provides the architecture to store and manipulating the objects**
* **Collection Framework contains group of I/f and implementation classes** 
  1. **sort array list of employee objects based on id**
* **Collections.sort(listofcountries,(e1,e2)->e1.id.compareTo(e2.id));**
* **Collections.sort(a,(a1,a2)->a1.getDxConceptId().compareTo(a2.getDxConceptId()));**

**Here (e1,e2)->e1.id.compareTo(e2.id)); is lambda expression. Which is providing implementation for compare abstract method of comparator Interface.**

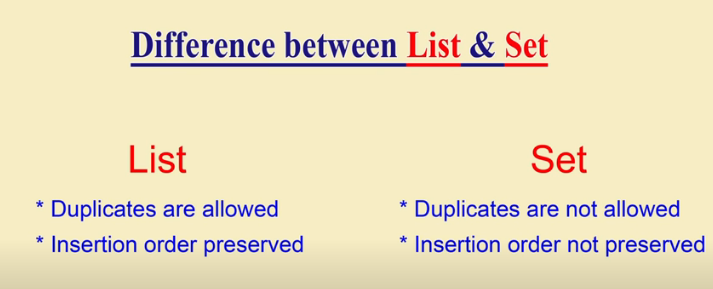
* **to customize sorting we can use comparator interface**
* **create sorting class which implements Comparator**
* **in sorting class override compare(a,b) to return sorting order**
* **sort the array list using Collection.sort(array, new sortingObj())**

**Ref:** [**https://www.geeksforgeeks.org/collections-sort-java-examples/**](https://www.geeksforgeeks.org/collections-sort-java-examples/)

* 1. **how to sort array list**
* **to sort the values in ascending order Collections.sort(list);**
* **if we want to sort in descending order reverse the sorted list like list.revers()**

**Ref:** [**https://www.geeksforgeeks.org/collections-sort-java-examples/**](https://www.geeksforgeeks.org/collections-sort-java-examples/)

* 1. **what will happen if we store array list into set**
* **if we store array list into hashset insertion order will not maintain, inserted based on hashCode value and duplicates removed**
* **if it is linked hashset insertion order will maintain but duplicates are removed**
* **if it is tree set insertion order will not maintain, inserted based on default sorting order(asc for numbers,alpha) and duplicates removed**
  1. **Difference between list and set**

****

* 1. **what is linked hash set**
* **LinkedHashSet internally uses LinkedHashMap to store objects**
* **It is the child class of hashset**
* **Here duplicates are not allowed**
* **Insertion order preserved**
  1. **Difference between hash set and linked hashset**

|  |  |  |
| --- | --- | --- |
| **S.No** | **HashSet** | **Linked HashSet** |
| **1.** | **HashSet internally uses HashMap for storing objects** | **LinkedHashSet uses LinkedHashMap internally to store objects** |
| **2.** | **Insertion order not preserved** | **Insertion order preserved** |
| **3.** | **It requires less memory** | **It uses more memory. Since it is using linked list internally** |
| **4** | **introduced in v1.2** | **Introduced in v 1.4i** |

**7.What is the difference between hashmap and treemap**

|  |  |  |
| --- | --- | --- |
| **Basis** | **HashMap** | **TreeMap** |
| **Definition** | **Java HashMap is a hashtable based implementation of Map interface.** | **Java TreeMap is a Tree structure-based implementation of Map interface.** |
| **Interface Implements** | **HashMap implements Map, Cloneable, and Serializable interface.** | **TreeMap implements NavigableMap, Cloneable, and Serializable interface.** |
| **Null Keys/ Values** | **HashMap allows a single null key and multiple null values.** | **TreeMap does not allow null keys but can have multiple null values.** |
| **Homogeneous/ Heterogeneous** | **HashMap allows heterogeneous elements because it does not perform sorting on keys.** | **TreeMap allows homogeneous values as a key because of sorting.** |
| **Performance** | **HashMap is faster than TreeMap because it provides constant-time performance that is O(1) for the basic operations like get() and put().** | **TreeMap is slow in comparison to HashMap because it provides the performance of O(log(n)) for most operations like add(), remove() and contains().** |
| **Data Structure** | **The HashMap table.** | **TreeMap internally uses a Red-Black tree, which is a self-balancing Binary Search Tree.** |
| **Comparison Method** | **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.** |
| **Functionality** | **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().** |
| **Order of elements** | **HashMap does not maintain any order.** | **The elements are sorted in natural order (ascending).** |
| **Uses** | **The HashMap should be used when we do not require key-value pair in sorted order.** | **The TreeMap should be used when we require key-value pair in sorted (ascending) order.** |

* 1. **Difference between Hashmap and weekhashmap**
* **in hashmap even object doesn't have any reference and associated with only hashmap, that object doesn't eligible for GC.**
* **HM is stronger(dominates) than GC**
* **but in the case of Week hashmap if object doesn't contain any reference it is eligible for GC, even though object associated with week hash map.**
* **i.e GC dominates week hash map**

**9. Difference between HashMap and HashSet?**

|  |  |  |
| --- | --- | --- |
| **Basis** | **HashMap** | **HashSet** |
| **Definition** | **Java HashMap is a hash table based implementation of Map interface.** | **HashSet is a Set. It creates a collection that uses a hash table for storage.** |
| **Implementation** | **HashMap implements Map, Cloneable, and Serializable interface es.** | **HashSet implements Set, Cloneable, Serializable, Iterable and Collection interfaces.** |
| **Stores** | **In HashMap we store a key-value pair. It maintains the mapping of key and value.** | **In HashSet, we store objects.** |
| **Duplicate values** | **It does not allow duplicate keys, but duplicate values are allowed.** | **It does not allow duplicate values.** |
| **Null values** | **It can contain a single null key and multiple null values.** | **It can contain a single null value.** |
| **Method of insertion** | **HashMap uses the put() method to add the elements in the HashMap.** | **HashSet uses the add() method to add elements in the HashSet.** |
| **Performance** | **HashMap is faster/ than HashSet because values are associated with a unique key.** | **HashSet is slower than HashMap because the member object is used for calculating hashcode value, which can be same for two objects.** |
| **The Number of objects** | **Only one object is created during the add operation.** | **There are two objects created during put operation, one for key and one for value.** |
| **Storing Mechanism** | **HashMap internally uses hashing to store objects.** | **HashSet internally uses a HashMap object to store objects.** |
| **Uses** | **Always prefer when we do not maintain the uniqueness.** | **It is used when we need to maintain the uniqueness of data.** |
| **Example** | **{a->4, b->9, c->5} Where a, b, c are keys and 4, 9, 5 are values associated with key.** |  |

**10.how you will iterate the map/ How to retrieve data from hashmap**

* **we cannot iterate a Map directly using**[**iterators**](https://www.geeksforgeeks.org/iterators-in-java/)**, because Map are not**[**Collection.**](https://www.geeksforgeeks.org/collections-in-java-2/)

**1.iterate over entry set**

* **Map.entrySet() method returns a collection-view(Set<Map.Entry<K, V>>) of the mappings contained in this map.**
* **So we can iterate over key-value pair using getKey() and getValue() methods of**[**Map.Entry<K, V>**](https://docs.oracle.com/javase/7/docs/api/java/util/Map.Entry.html)**.**

**Map<String,String> gfg = new HashMap<String,String>();**

**// enter name/url pair**

**gfg.put("GFG", "geeksforgeeks.org");**

**gfg.put("Practice", "practice.geeksforgeeks.org");**

**gfg.put("Code", "code.geeksforgeeks.org");**

**gfg.put("Quiz", "quiz.geeksforgeeks.org");**

**// using for-each loop for iteration over Map.entrySet()**

**for (Map.Entry<String,String> entry : gfg.entrySet())**

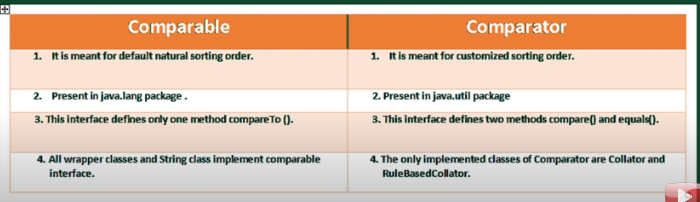
**System.out.println("Key = " + entry.getKey() +**

**", Value = " + entry.getValue());**

* 1. **Iterating over keys or values using keySet() and values() methods**

[**https://www.geeksforgeeks.org/iterate-map-java/**](https://www.geeksforgeeks.org/iterate-map-java/)

**11.Difference b/w comparator and comparable**

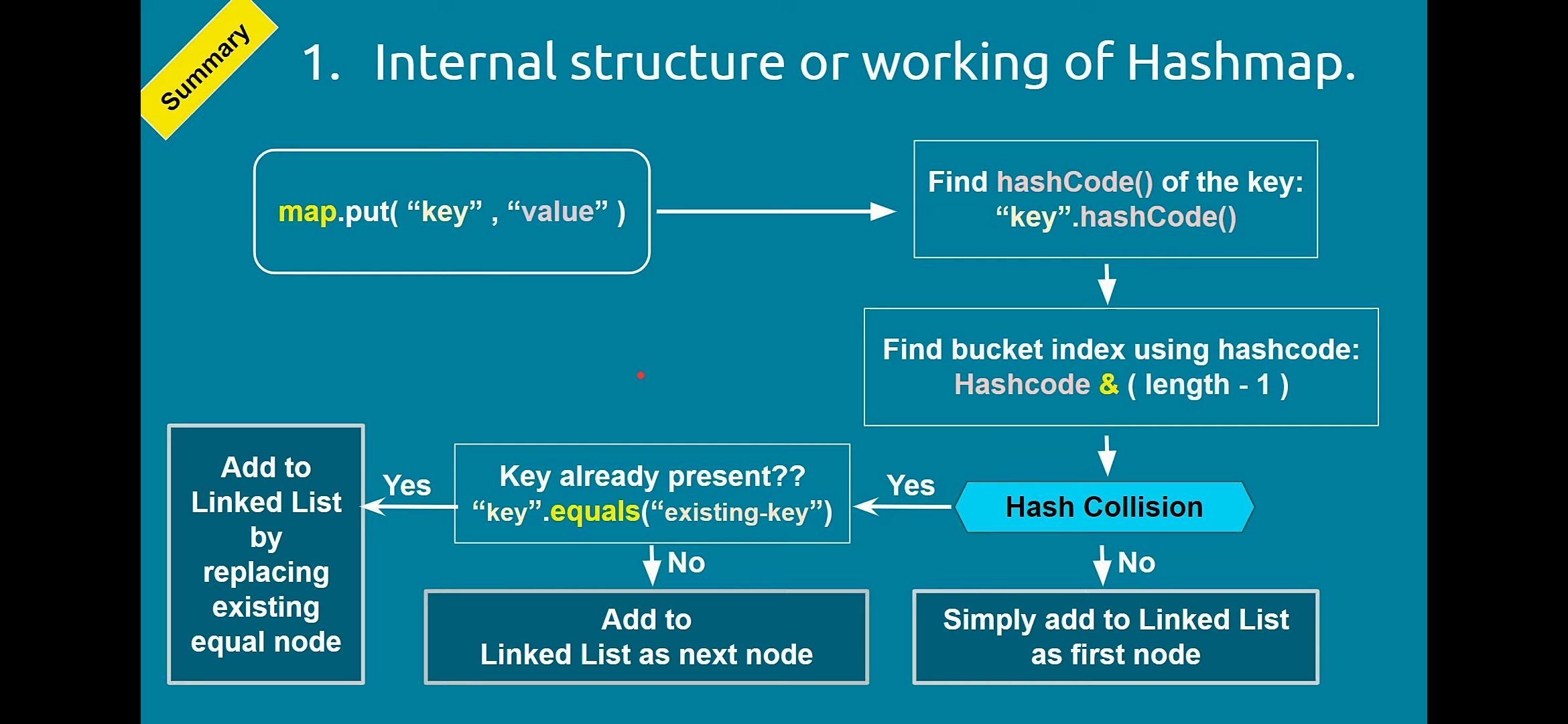
****

* **Both are functional interface.**
* **Comparable – compareTo() – abstract method**
* **Comparator – compare() – abstract method**

1. **When you will use the collection**
   1. **Array list – when we want to have duplicates and null values and insertion order. And if our frequent operation search/fetch**
   2. **Linked list – when we want to have duplicates and null values and insertion order. And it is used in manipulating collection objects.**
   3. **HashSet – when we want to store the data which is not accepting duplicates and insertion order is does not care**
   4. **Tree set - when we want to store the data which is not accepting duplicates and insertion order is does not care. and should not allow null values.**
2. **Different between array list and vector**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| **1) ArrayList is not synchronized.** | **Vector is synchronized.** |
| **2) 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.** |
| **3) ArrayList is not a legacy class. It is introduced in JDK 1.2.** | **Vector is a legacy class.** |
| **4) ArrayList is fast because it is non-synchronized.** | **Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.** |
| **5) ArrayList uses the Iterator interface to traverse the elements.** | **A Vector can use the Iterator interface or Enumeration interface to traverse the elements.** |

**14.how HashMap is working internally?**

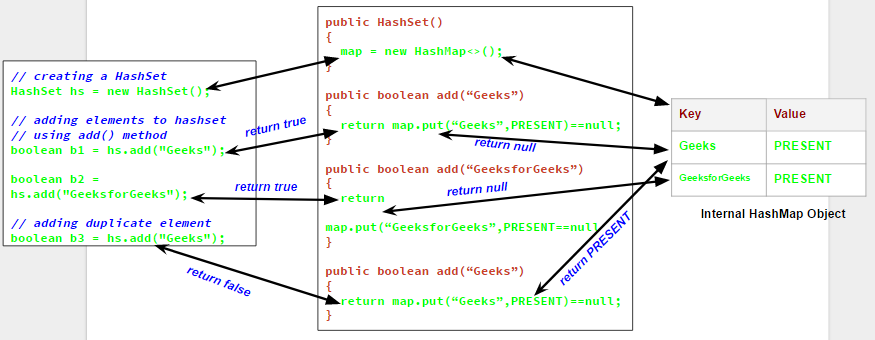
****

* **Internal data structure for hashmap is hashtable**
* **map.put(key,value) method is used to place the key-value pair entry in hashmap**
* **Here hashtable contains the list of buckets, initial bucket size is 16**
* **In each bucket we have a linked list to store more than 1 node**
* **Each node contain, key, value, and address of next node.**
* **When we call the put method, JVM will find the hashcode (integer form of key address/reference) of key**
* **Using hash technique jvm will find the bucket index to store the map entry.**
* **If bucket is empty, hash collision will not occur or else hash collision will occur**
* **If bucket is empty entry will be added to Linked List as first node.**
* **If bucket is not empty, using equals method new key will be compared against existing key value to check the key already exist or not.**
* **This step will be repeated until the linked list next node is null**
* **If key is not exist and linked list next node address is null new key value pair node will be added to linked list**
* **If key is exist, the value will be replaced with new value and returns old value.**
* **It won’t throw any error.**
* **Since it needs to search key in linked list the performance is low for search operation**
* **Hence hashmap performance getting slow for long linked list**
* **To overcome this in JAVA 8 hashmap enhanced to improve performance.**
* **The idea is, linked list will be replaced by binary search tree when it is reached the certain threshold. the threshold is known as “ Treeify threshold “, And the limit is 8.**
* **In binary search tree compareTo() is used to compare the key to check whether the given key is less or greater then existing key.**
* **If we try to add null in hashmap , since hashcode of null is 0, so null key stored in index 0,**
* **If we add again one more null key again it will be added to same index 0 that time the value will be replaced duplication null key not added**

**15. What is hash collision, how to resolve it / What will happen if value added for existing key**

* **In hash map while adding the key value using put method. First it will find the bucket index using hash value**
* **If index is same for two key that time hash Collison will happen.**
* **In that case in same bucket we have to add one more node**
* **HashMap always contains unique keys. If same key is inserted again, the new object replaces the previous object.**

**16. How hash set work internally? /How duplicates are identified in hashset**

****

* **Hash set is internally using hashmap to store elements**
* **When we add the value to the set, internally the value will be saved as key in hash map object and value for the corresponding key is “PRESENTT”**
* **It will return null if the key is new**
* **It will return PRESENT/old value if the key already present**
* **While adding the element to the set , set internally check whether the put method returns true or false**
* **when we try to add a duplicate element to a set using add() method, it returns false, and element is not added to hashset, as it is already present**

**17. What will happen if 2 user defined unique Employee obj added in hash set**

**Eg: FilterEmployee prgm in package** com.wm\_practice.collection;

* **When we try to add below employee obj in set**

Employee1 emp1= **new** Employee1("emp1", 10000, "HR");

Employee1 emp2 = **new** Employee1("emp1", 10000, "HR");

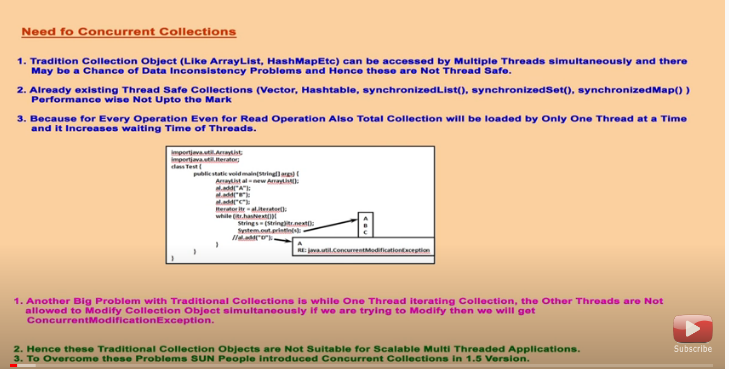
Set<Employee1> emp = **new** HashSet<>();

emp.add(emp1);

emp.add(emp2);

* **Here both emp1 and emp2 has same content logically,**
* **Set should not allow the duplicates but both emp1,emp2 added to set**
* **Bcz emp1 and emp2 has different hashcode value or memory address ,**
* **As per the contract between hashcode and equals method if hashcode is diff for 2 obj then both are never equal,**
* **Since employee class does not override hashcode / equals method object class equals method invoked, which means emp1 and emp2 reference are not equal hence duplicate added to set**
* **To resolve this we need to override hashcode and equals method to define how to determine uniqueness of employee obj/user defined obj. like entire ob should be unique or emp name alone unique or id alone unique**

**18.concurrent collections**

****

**19.Different between hash map and hash table/what will happen if we use hash table instead of hash map.**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| **1) HashMap is non synchronized. It is not-thread safe.** | **Hashtable is synchronized. It is thread-safe..** |
| **2) HashMap allows one null key and multiple null values.** | **Hashtable doesn't allow any null key or value.** |
| **3) HashMap is a new class introduced in JDK 1.2.** | **Hashtable is a legacy class.** |
| **4) HashMap is fast.** | **Hashtable is slow.** |
| **5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);** | **Hashtable is internally synchronized and can't be unsynchronized.** |
| **6) HashMap is traversed by Iterator.** | **Hashtable is traversed by Enumerator and Iterator.** |
| **7) Iterator in HashMap is fail-fast.** | **Enumerator in Hashtable is not fail-fast.** |

**20. What is concurrent modification exception? How to avoid hashrrent modification exception**

**The ConcurrentModificationException occurs when an object is tried to be modified concurrently when it is not permissible. This exception usually comes when one is working with Java Collection classes.**

**For Example - It is not permissible for a thread to modify a Collection when some other thread is iterating over it.**

**Technique to avoid :**

1. **If you can’t stop the underlying collection from being modified during iteration, create a clone of the target data structure and iterate through the clone**
2. **Use a fail-safe collection class that will not throw the ConcurrentModificationException**
3. **We can modify the iterator object structure instead of collection**

**21.What is fail fast and fail safe iterators**

| **Sr. No.** | **Key** | **Fail-Fast** | **Fail-Safe** |
| --- | --- | --- | --- |
| **1** | **Exception** | **Any changes in the collection, such as adding, removing and updating collection during a thread are iterating collection then Fail fast throw concurrent modification exception.** | **The fail-safe collection doesn't throw exception.** |
| **2.** | **Type of collection** | **ArrayList and hashmap collection are the examples of fail-fast iterator** | **CopyOnWrite and concurrenthashmap are the examples of a fail-safe iterator** |
| **3.** | **Performance and Memory** | **It's work on actual collection instead. So, this iterator doesn't require extra memory and time** | **It's working on a clone of the collection instead of actual collection. It is overhead in terms of time and memory** |
| **4.** | **Modifications** | **Iterators don't allow modifications of a collection while iterating over it.** | **Fail-Safe iterators allow modifications of a collection while iterating over it.** |

**22.What is conurrent hashmap**

* **it is the implementation class of ConcurrentMap**
* **underlying architecture is hashtable**
* **in hashtable, at a time only one thread allowed to operate on collection object. even read operation. hence it is thread safe.**
* **in concurrent hash map, if it is read operation multiple threads allowed to read. no restriction and no waiting time. Hence it is allowed concurrent read operations**
* **if it is update/write operation instead of locking total collection object(in hash table) we are locking bucket level. thread safe write operation allowed**
* **null is not allowed both key/value operation**
* **hence while one thread reading object if other thread modified the collection it won't throw concurrent modification exception**

**23.Difference between collection and stream API, which performance is high**

**24. Internal implementation for push and pop in Stack**

**25. Diff b/w collection and collections**

**Java 8 Features**

**1.Java 8 features**

**The main purpose of java 8 is to write concise, readable and more maintainable code.**

* 1. **lambda expression**
  2. **functional interface**
  3. **default methods in interfaces**
  4. **static methods in interfaces**
  5. **streamAPI**
  6. **method reference(::)**
  7. **Optional**
  8. **Predicate**
  9. **Function**
  10. **Consumer**
  11. **Supplier**
  12. **forEach()**
  13. **Collect**

1. **what is functional interfaces, what are the predefined functional interfaces available In java**

**Why functional interface:**

* **To call lambda expression we have to use functional interface**
* **Without FI we can’t use lambda expression**

**What Is functional interface**

* **An interface contains single abstract method known as Functional interface**
* **FI has any number of default, static method but only one abstract method**

**Runnable, - run()**

**Comparable – compareTo()**

**Comparator -compare() Also FI bcz it contains only one abstract method.**

* **Predefined FI available in java.utill.function package**
* **These predefined/ build in FI are used in lambda expression, stream API**

**Predefined functional interface**

* + - 1. **Predicate**
      2. **Function**
      3. **Consumer**
      4. **Supplier**

**Two arguments Predefined functional interfaces**

* + - 1. **BiPredicate**
      2. **BiFunction**
      3. **BiConsumer**

1. **What is default method, why we should have default method?**

* **if we want to add method in interface without impact child go for default method**
* **default method created in Functional interface with Default keyword and method body**
* **It is used to provide common implementations for all the child classes and can override default method.**
* **we can’t create method with default keyword in class. So if we want to override default method we have to use public/protected keyword for the method**
* **in implementation class if we want parent class implementation, if both parent class same method we can override the method In child class and call the parent implementation inside that method ,Left.super.m1() – interface\_name.super.method\_name**

1. **what is use of static method in interface (Java 8)?**

* **Static method called by using class name itself**
* **If we are only having static method in classes no concreate method , then we can go for Interface**
* **Bcz interface is light weight and we don’t need to create object**
* **Interface’s static method by default not available to implementation class**
* **We have to call static method using interface name only like, interf.method\_name();**
* **We can have main method in interface also**

1. **What is Stream api? filter odd employee id from employee objects**

* **Stream api is used to process the collection of objects**
* **Stream provide the result without changing the original collection object**
* **Convert collection to stream**

**Collection<String> collection = Arrays.asList("a", "b", "c");**

**Stream<String> streamOfCollection = collection.stream();**

**Convert array to stream**

**Stream<String> s =Stream.of(“a”,”b”,”c”);**

**String[] arr = new String[]{"a", "b", "c"};**

**Stream<String> streamOfArrayFull = Arrays.stream(arr);**

**Stream<String> streamOfArrayPart = Arrays.stream(arr, 1, 3);**

**ArrayList al = new ArrayList()**

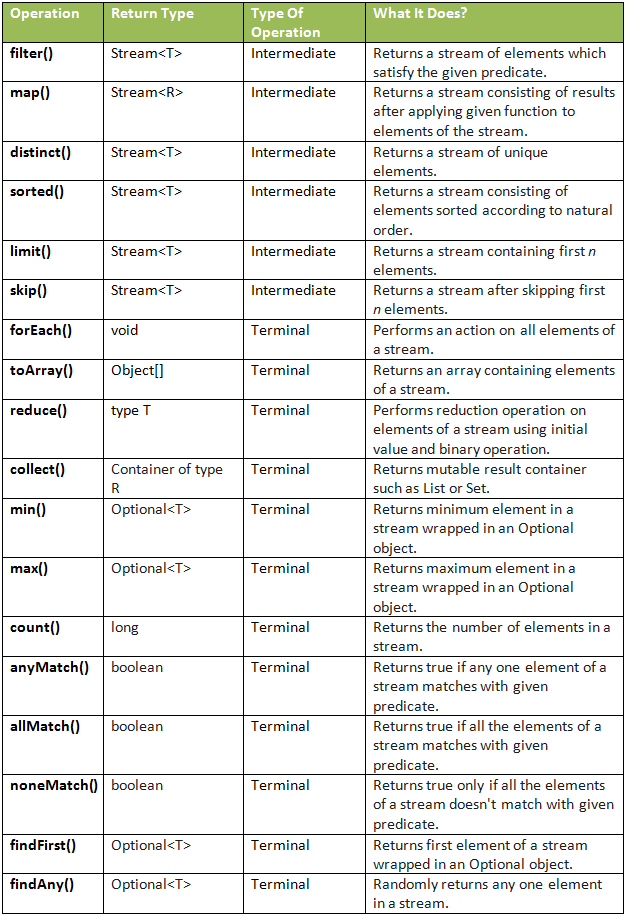
**//added employee objects into al**

**//a l.add(new Employee(1,”Maya”,2000))**

**ArrayList emp= al.stream().filter((e)->e.id%2!=0).collect(Collectors.toList());**

1. **What is intermediate and terminal operations in Java 8**

|  |  |
| --- | --- |
| **Intermediate Operations** | **Terminal Operations** |
| **They return stream.** | **They return non-stream values. Like collection, string, primitive** |
| **They can be chained together to form a pipeline of operations.** | **They can’t be chained together.** |
| **Pipeline of operations may contain any number of intermediate operations.** | **can have maximum one terminal operation, that too at the end.** |
| **Intermediate operations are lazily loaded.bcz it Is executed at the time of terminal operation called** | **Terminal operations are eagerly loaded.** |
| **They don’t produce end result.** | **They produce end result.** |
| **Examples : filter(), map(), distinct(), sorted(), limit(), skip()** | **Examples : forEach(), toArray(), reduce(), collect(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()** |

****

**19. what is forEach in java 8?**

* **with forEach, we can iterate over a collection,stream objects and perform a given action on each element, like any other Iterator.**
* It is defined in Iterable and Stream interface.
* It is a default method defined in the Iterable interface.
* Collection classes which extends Iterable interface can use forEach loop to iterate elements.
* This method takes a single parameter which is a Consumer functional interface. So, you can pass lambda expression as an argument.

**For instance, consider a for-loop version of iterating and printing a Collection of Strings:**

**for (String name : names) {**

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

**}**

**We can write this using forEach:**

**names.forEach(name -> {**

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

**});**

**Inside forEach We are using Consumer Functional interface**

**20.optional class in java8? –prgrm required for methods**

* **Optional class introduced in java 8 to handle null values.**

**Advantage of Optional Class:**

* **To avoid null point exception we need to do many null check, if we missed any one leads to get null check.**
* **To overcome this java introduced Optional class , it is final class hence not eligible for inheritance**
* It can help in writing a neat code without using too many null checks.
* **Clean code & easy to read and understand**
* Optional is a container object which may or may not contain a non-null value.

**Eg:**

String[] words = new String[10];

        Optional<String> checkNull

            = Optional.ofNullable(words[5]);

        if (checkNull.isPresent()) {

            String word = words[5].toLowerCase();

            System.out.print(word);

        }

        else

            System.out.println("word is null");

    }

**Methods:**

* **Optional.of(T value) – returns the optional with specified non null value**
* **Optional.ofNullable- returns an optional specified non null value or else return empty optional**
* **Optional.empty – return empty optional**
* **Get() – if value present return the value, or else throws NoSuchElementException**
* **ifPresent(Consumer)- if the value is present invoke the consumer with the specified value otherwise do nothing,**
* **isPresent() - Returns true if there is value present or else false**
* **map(Function)**
* **flatMap(Function)**
* **filter(Predicate)**
* **orElse()**
* **orElseGet()**
* **orElseThrow()**

**Ref: https://www.geeksforgeeks.org/java-8-optional-class/**

**21.What is method reference?**

* **A method reference is the shorthand syntax for a lambda expression that executes just ONE method. Here's the general syntax of a method reference:**
* **Object :: methodName**
* **Eg: a::m1, system.out ::println, String::valueOf, Integer::parseInt**

**22. what are the java 8 features you have used in your projects**

* **Lambda expression – to sort the arraylist using comparator**
* **Method reference**
* **Stream api - to filter /map/ sort the arraylist using comparator**
* **Functional interface**
* **Predicate**
* **Function**
* **consumer**
* **forEach()**
* **collect**

**23. Collect method in streams?**

* **This Stream method is a terminal operation which reads given stream and returns a value or collection like List or Set (or Map)**
* **collect() method accepts Collectors which is a final class with various utility methods to perform reduction on the given Stream**
* **To be very precise, we are going to discuss only 4 methods namely  
  1. Collectors.toList() to convert to List  
  2. Collectors.toSet() to convert to Set  
  3. Collectors.toCollection() to convert any Collection class like ArrayList/HashSet  
  4. Collectors.toMap() to convert to Map**

**5.Collectors.groupingBy() to group the collection object**

**24. what is Flatmap**

* **Java 8 stream API provides map() and flatMap() method, Both these methods are intermediate methods and returns another stream as part of the output.**
* **Map() method used for transformation. Also its mapper function produces single value for each input value. hence it is also called as One-to-One mapping.**
* **flatMap() used for transformation & flattering. i.e flatMap() = map()+flattering**
* **its mapper function produces the multiple value for each input value. Hence it is also called as one-to -many-mapping**
* **The Stream.map() method performs an intermediate operation by using the mapper function. It produces a new stream for each element. It transforms all the streams into a single stream to provide the result. therefore, each element of the stream gets converted into a new stream.**
* **Stream.flatMap() is used to convert a Stream of Stream into a list of values using flattening mechanism**
* **Flattening is the process of converting several lists of lists and merge all those lists to create a single list containing all the elements from all the lists.**

**Eg:**

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

**{**

**//creating ArrayList**

**List<String> productlist1 = Arrays.asList("Printer", "Mouse", "Keyboard", "Motherboard");**

**List<String>  productlist2 = Arrays.asList("Scanner", "Projector", "Light Pen");**

**List<String> productlist3 = Arrays.asList("Pen Drive", "Charger", "WIFI Adapter", "Cooling Fan");**

**List<String> productlist4 = Arrays.asList("CPU Cabinet", "WebCam", "USB Light", "Microphone", "Power cable");**

**List<List<String>> allproducts = new ArrayList**

**<List<String>>();**

**//adding elements to the list**

**allproducts.add(productlist1);**

**allproducts.add(productlist2);**

**allproducts.add(productlist3);**

**allproducts.add(productlist4);**

**//creating a list of all products**

**List<String> listOfAllProducts = new ArrayList<String>();**

**//for each loop iterates over the list**

**for(List<String> pro : allproducts)**

**{**

**for(String product : pro)**

**{**

**//adds all products to the list**

**listOfAllProducts.add(product);**

**}**

**}**

**System.out.println("List Before Applying mapping and Flattening: \n");**

**//prints stream before applying the flatMap() method**

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

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

**//creats a stream of elemnts using flatMap()**

**List<String> flatMapList = allproducts .stream().flatMap(pList -> pList.stream()).collect(Collectors.toList());**

**System.out.println("List After Applying Mapping and Flattening Operation: \n");**

**//prints the new stream that we get after applying mapping and flattening**

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

**}**

**}**

**Output:**

**List Before Applying mapping and Flattening:**

**[Printer, Mouse, Keyboard, Motherboard, Scanner, Projector, Lighten, Pen Drive, Charger, WIFI Adapter, Cooling Fan, CPU Cabinet, WebCam, USB Light, Microphone]**

**List After Applying Mapping and Flattening Operation:**

**[Printer, Mouse, Keyboard, Motherboard, Scanner, Projector, Light Pen, Pen Drive, Charger, WIFI Adapter, Cooling Fan, CPU Cabinet, WebCam, USB Light, Microphone]**

**25. Diff b/w Map , FlatMap**

|  |  |
| --- | --- |
| **Stream.flatMap()** | **Stream.map()** |
| **It processes the stream of stream's values.** | **It processes the stream of values.** |
| **It performs mapping along with flattening.** | **It performs mapping only.** |
| **It transforms data from Stream<stream> to Stream.** | **It transforms data from Stream to Stream.** |
| **It uses One-To-Many mapping.** | **It uses One-To-One mapping.** |
| **It's mapper function produces multiple values (stream of values) for each input value.** | **It's mapper function produces single values for each input value.** |
| **Use the flatMap() method when the mapper function is producing multiple values for each input value.** | **Use the map() method when the mapper function is producing single values for each input value.** |

**26. Types of Stream**

**1.Sequential Stream:**

* **Sequential Streams are non-parallel streams that use a single thread to process the pipelining.**
* **Any stream operation without explicitly specified as parallel is treated as a sequential stream.**
* **stream() method returns a sequential stream in Java.**
  1. **Parallel Stream**
* **It is a very useful feature of Java to use parallel processing, even if the whole program may not be parallelized.**
* **Parallel stream leverage multi-core processors, which increases its performance.**
* **Using parallel streams, our code gets divide into multiple streams which can be executed parallelly on separate cores of the system and the final result is shown as the combination of all the individual core’s outcomes.**
* **Eg:list.parallelStream().forEach(System.out::print);**

**27.Predicate Interface**

* **It is predefined/build in functional interface**
* **It used to perform Boolean/conditional check**
* **It is part of java.utill.function package**
* **It has abstract method called test**
* **Which accepts one argument and returns Boolean value**
* **Passed as an argument in Filter method of Stream api , to perform filter operation using conditional check**

Predicate<Employee> salary = e -> e.getSalary()>20000

List<Employee> maleEmployeeList = employeeList.stream().filter(salary).toList();

**28.Consumer Interface**

* **It is Functional Interface part of java.utill.function package**
* **Used to enable functional programming in java**
* **It has accept(Object) abstract method which will accept single input , perform some operation and returns no output,**
* **Consumer Interface passed as argument in forEach() method**

**Eg:** Consumer<String> printConsumer = name -> System.out.println(name);

Names.forEach(printConsumer);

**29. Supplier**

* **It is predefined built in functional interface**
* **Supplier is used to supply some value**
* **It is part of java.utill.function package**
* **It represent single function get(), which does not take in any argument and produce value of type T**
* **Supplier accept one input type which is the output return type**
* The lambda expression assigned to an object of Supplier type is used to define its **get()** which eventually produces a value

**Eg:** // This function returns a random value.

        Supplier<Double> randomValue = () -> Math.random();

        // Print the random value using get()

        System.out.println(randomValue.get());

**30.Function**

* **Functions also a predefined / built in functional interface**
* **It is part of java.utill.function package**
* **Function has abstract method called apply(Object), which take some input, perform some operations and returns some results**
* **Function passed as argument in Stream API ‘s Map method.**

**Eg:**  Function<Integer, Double> half = a -> a / 2.0;

        // Applying the function to get the result

        System.out.println(half.apply(10));

**31. Which one the more efficient forEach() loop or Stream().forEach()?**

* **Collection.forEach is better & faster ,**
* **Because in stream we need to convert collection into stream which will take extra time and memory space**

**30.why we need functional programming in java**

**31.JVM optimization in java8**

**32. what is completable feature**

**33. Collection vs stream, which is best to use**

**34. Java8 Date API**

**35. Exception handling in lambda expression – need video tutorial**

**36. Collectors . partitionBy**

* + - 1. **SOLID principle**
      2. **What is volatile,value saved using volatile can available for other thread, is volatile and static are same**
      3. **Why wait, notify methods defined in Object class**
      4. **Diff between runnable & callable?- what/why callable – video required**
      5. **What are thye default attributes of @Transactional**
      6. **What is Autowiring and types of auto wiring**
      7. **Global exception handler, how the exception comes to here**
      8. **What is @Advice, @ Control advice**
      9. **Executor framework**
      10. **How to handle transaction failures between 2 microservice / Saga design pattern**
      11. **If we changed client endpoint do we need to restart Eureka server - no chk once**
      12. **What is token? What r the different part in Token**
      13. **Diff b/w outh2 and jwt**
      14. **Have you implemented single sign on in ur application**
      15. **How to communicate between 2 microservice asynchronously – webclient/ kafka mq**
      16. **Diff between sync and async communication between 2 micro service**
      17. **Why we are using ResponseEntity**
      18. **What is SAGA design pattern**
      19. **What is Centralized logging mechanism**
      20. How to stream the List<List<String>list; -flatmap syntax
      21. **Streams Time and space complexity or optimization?**
      22. **Parallel Streams: How many threads will be created and how does it distribute our collections internally, Can we define our logic for data separation**
      23. What is disposable princible in micoservice
      24. What are the 12 microservice principles
      25. Inner static class in builder pattern
      26. sort the employee list based on salary in descending order using java 8 – Solution : package : com.wm\_practice.collection, class **: SortEmployee.java**