* **Basic Java:**

1. Explain important features of java?

* Java is platform Independent Programming Language.
* It is pure Object Oriented Programming Language.

(But not 100% because – Java use **Static** keyword.

– Java class doesn’t support multiple inheritances.

– Java still use primitive data members (int, float, etc.)

* It is open source language. (Software, Documentation and source code are freely available.)

1. **What is class?**

* It is different type of entities, items, (Ex. Table, Ball Pens, Mobiles, etc.)
* It is one type of templates, blueprint or prototype. (Ex. College Examform, Aadhar Card Form, etc.)
* Syntax:

public Class A

{

}

public = Access Modifier

Class = Keyword

A = Class Name

* It’s body contains methods, objects, variables, constructors, blocks.

1. **What is main method?**

* Syntax:

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

-public =Access modifier

-static =keyword

-void =Nothing return

-main =Method name

-String[] =String array

-args =Local variable name

1. **What is JVM?**

* JVM is virtual machin that runs the bytecode.

1. **What is JRE?**

* JRE consist of libraries (jar) and JVM to run application written in java.
* Just to run application you only need JRE.

1. **What is JDK?**

* JDK is super set of JRE.
* It contains everything that JRE along with development tools for developing java application, debugging tools for monitoring java application.

1. **Explain System.out.println(“”);**

* System = class
* out=object of ‘Printstream’ class
* println=method of ‘Printstream’class

1. **Why ‘main’ method is public?**

* It can access from anywhere.

1. **Why ‘main’ method is static?**

* JVM call main method by class name only.

1. **Why ‘main’ method’s return type is ‘void’?**

* JVM doesn’t want any return value from ‘main’ method.

1. **What is use of (Strings[] args) in ‘main’ method?**

* To take input from Command Line.

1. **Why it is ‘String Array’ only?**

* ‘String’ can hold any java type of value.

1. **What is package?**

* ‘package’ is nothing but folder.
* It is a group of similar purpose .class file.

1. **How to compile package source file?**

* javac -d . A.java (-d =make a directory

. =current location

A.java=source file)

1. **How to run compiled package file?**

* Using it’s full qualified name.
* com.yash.bank.A

1. **What is object?**

* Object is an instance of class.
* It is separate copy of memory.
* It has state and behavior.
* State can be change object to object.
* Behavior can’t be change object to object.
* Eg: Dog has state: color, name and Behavior: eat , smell
* State: represent the data(value) of an object.
* Behavior: represent behavior(functionality) of an object
* Identity

1. **What are static blocks and static initializers in Java ?**

* Static blocks or static initializers are used to initalize static fields in java.
* We declare static block when we want to initialize static field in our class.
* Static blocks gets executed exactly once when the class is loaded.
* Static blocks are executed even before the constructors are executed.

1. **contract between hashcode and equals?**

* The basic rule of the contract states that if two objects are equal to each other based on equals() method, then the hash code must be the same, but if the hash code is the same, then equals() can return false.

**Oops:**

1. **What is Abstraction?**

* Hiding internal implementation and sharing set of services is called as abstarction.
* We can achieve abstraction by using “Interface” and “Abstarct Class”.
* We can achieve 100% abstraction using “Interface” and partial by using “Abstract Class”.
* We can achieve security using abstraction.
* There are several place where we need to use abstraction:

-while development of project structure and layout.

-for creating API we will use abstraction.

- third party communication we will use abstraction(eg. IRCTC and Bank).

* E.g. ATM machine, Car, Mobile, etc.

1. **What is API?**

* Application programming interface
* It provide rules, guidelines and specification.
* Through API uniform will be maintain.
* There are 3 different types of API:

1. Inbuild API: Rules and guidelines provided by java and there implementation is also provided by java. (eg: Multithreading , collection)
2. Open specification API: Rules and guidelines provided by java and there implementation is written by different vendor (eg: JDBC, JSP)
3. Third party API: Rules and guidelines provided by different vendor and there implementation is also written by them. (eg: spring, Hibernate)
4. **What is interface?**

* Interface is set of rules.
* In interface all variable are by-default **public**, **final** and **static**.
* In interface all methods are ended with semicolon (;).
* It is by-default **public** and **abstract** only. e.g. public abstract void m2( ) ; till jdk 1.7
* After 1.8 we can write static and default method in interface.
* In 1.9 it support private also.
* We cannot create object of interface. Because method are incomplete and there is no any constructor available in interface. Because variable are public static final that why no constructor is there.
* We can create reference of interface by using there implemented class.
* Class implement interface by using implement keyword.
* Eg :

public interfacr I

{

void m1();

}

public class A implements I

{

Public void m1()

{

Syso(“m1 of A”);

}

public static void main(String args[])

{

I i = new A();

i.m1();

}

}

1. **Why Java class doesn’t support multiple inheritance?**

* There is chances of ambiguity , for avoiding that ambiguity java does not support multiple inheritance.

1. **What is marker interface?**

* It is also called “tag” interface.
* Interface which has blank body i.e. called “marker” or “tag” interface.
* It is used for decision making purpose.
* JVM will take decision through marker interface.
* There are several marker interface given by Sunmicrosystem. e.g. Serialisable, Clonable, Remote etc.

1. **What is Abstract class?**

* It is called partial class.
* Abstract class have both type of method implemented as well as Abstract(unimplemented).
* If class have anyone method is “abstract”, then class must be “abstract”
* Syntax :-

abstract public class A

{

public void m1( )

{

}

abstract void m1( ) ;

}

* If class have no abstract method, then also we can make class as a abstract.

1. **Why we make class as abstract, if class have no abstract methods?**

* No one can create object of that class.

1. **How we can call abstract class implemented method?**

* Through inheritance, through sub-class.

**public abstract class** B **public class** C **extends** B

{ {

**public void** m1() **void** m4()

{ {

} m1();

m2();

**public void** m2() }

{ }

}

**public class** Test

{

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

{

B b=**new** C();

b.m1();

b.m2();

C c=**new** C();

c.m1();

c.m2();

c.m4();

}

}

1. **How we can create child class of abstract class which have both type of method?**

* Child class must implement all abstract method of abstract class.
* If child class is unable to implement all abstract method of abstract class then make that child class as abstract.

1. **Is there any constructor inside abstract class? If yes, what is use of it?**

* Yes, to initialise variables of that class.

1. **What is access modifier of abstract class constructor?**

* by default it is “public”.

1. **What is Encapsulation?**

* Wrapping of data member and methods called as Encapsulation.
* We can achieve it by making data members “private”.
* POJO class is good example of encapsulation.
* In a class if it has every data member as a “private” then such class is called as tightly encapsulated.
* We achieve data hiding concept through encapsulation.
* Eg: Engine, Gear box within car.
* Public : Access anywhere
* Protected: Access within package and outside the package should be subclass.
* Default: Access within package.
* Private: Access within class

1. **Difference between Abstraction and Encapsulation?**

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| Hiding internal implementation and sharing set of services. | Wrapping of data member and methods. |
| We can achieve abstraction by using “interface” and “Abstract class” keyword. | We can achieve it by making data member as “private”. |
| It increased code | It decreases code |
| It solve problem at Design level | It solve problem at implementation level. |

1. **What is inheritance?**

* Acquiring properties of parent class is called inheritance.
* It is also called as IS-A relationship.
* It can be achieved by using “extends” keyword, by making Parent-Child relationship.
* The purpose of inheritance is reusability of code.
* Java class does not support multiple inheritance so that hybrid is not possible.
* Eg: New car version inherits properties from old version.
* Types of Inheritance: Single level, Multilevel, Hierarchical, Multiple and Hybrid (java does not support).
* Subclass inherit all the member of superclass
* Constructor are not member so they are not inherit by subclass. But constructor we can invoke from subclass.

1. **Return type in Inheritance?**

* If method return type is class type then we can written same class return or there child type class of return .
* But we cannot return parent type of reference.

**public class** A **public class** B **extends** A **public class** C **extends** B

{ { {

} } }

**public class** Test

{

**public** A m1()

{

A a=**new** A();

B b=**new** B();

C c=**new** C();

//return a;

//return b;

**return** c;

}

**public** B m2()

{

A a=**new** A();

B b=**new** B();

C c=**new** C();

//return b;

**return** c;

}

**public** C m3()

{

A a=**new** A();

B b=**new** B();

C c=**new** C();

**return** c;

}

}

1. **Super and this keyword?**

* Both are Non-Static keyword.
* It is used for method calling, constructor calling and varible calling.
* “super” keyword is by default first line of every constructor.
* “super” keyword in constructor is used for calling Parent class constructor.
* “super” keyword we can write explicitly any of the constructor but it should be the first line.
* Every class must have at least one constructor with “super” keyword.
* We can write “this” keyword in constructor’s first line instead of “super” keyword.
* “this” keyword is used for calling same class another constructor.
* We can write (n-1) “this” keyword in class (n = Number of Constructor).
* “this” and “super” both at a time cannot write in single constructor for calling another constructor.
* “this” we need to write explicitly, but it should be first line.

1. **What is polymorphism?**

* It means one name many forms.
* It makes reusability simple and also makes code understanding easy.
* Best example of polymorphism is “println” method of “printstream” class.
* There are two types of Polymorphism :
* Run-time Polymorphism (Dynamic Binding, Overriding)
  + Decision making at Runtime by runtime object.
  + Used for adding additional functionality into existing one.
  + Useful only in Parent-Child Relationship.
  + We cannot override **static, private, final** methods.
  + Best example of method overriding is ‘equals’ method of object class and ‘equals’ method of String class.
  + We can override method only not constructor.
* Syntax rule:
* Method name must be same
* Method parameter or signature must be same.
* If method return type is void or primitive then it must be same.
* If method return type is user-defined data member then return type of subclass method should be same
* Access modifier should be same or greater, it cannot be weaker. ( public > protected > default > private )
*  Private Method :-
  + If method is private then it will not be override.
  + Private method we can write as it is in their child class.
  + Private method will not be inherited.
  + Private method we cannot call outside the class.
*  Static Method :-
  + If method is static then it will not be override.
  + Static method we can write as it is in their child class but compulsory with ‘static’ keyword.
  + Static method will be inherited.
  + It can be call outside of the class through object and through class name.
*  Final Method :-
  + Final method, we cannot override.
  + It cannot be written as it is in their child class.
  + We can inherit final method.
  + It can be call outside of the class through object only.
* Compile-time Polymorphism (Static Binding, Overloading)
* Decision making at Compile time by reference object.
* Used for adding same functionality for different data inputs.
* We can overload method and constructor both.
* Constructor overloading only be happenin same class.
* Method overloading can be happen in same class or in between Parent-Child class.
* We can overload static, private and final method.
* Best example of method overloading is “wait” method of object class and “println” method of printstream class.
* Method overloading syntax rule:
* Method name must be same.
* Method parameter (signature) must be different.
* Return type and access modifier doesn’t matter.

1. **Why we call overloading as compile-time Polymorphism?**

* Every decision will happen only at compile time , that’s why it is called compile time polymorphism.

1. **What is hashcollision?**

* In HashMap, if two keys have same hashcodes then such situation is called as hash collision.
* In such case, while adding data, doubly Linkedlist is created to insert data.

1. **Difference between Abstract class and interface**

|  |  |
| --- | --- |
| Abstract class | Interface |
| Abstract class can have abstract and non abstract method | Interface can have only abstract methods. Since java 8 it can have default and static method also |
| Abstract class doesn’t support multiple inheritance | Interface support multiple inheritance |
| Abstract class can have final, non final, static and non static variable. | Interface has only static and final variables |
| Abstract class can provide the implementation of interface. | Interface can’t provide implementation of abstract class |
| The abstract keyword is used to declare abstract class | The interface keyword is used to declare interface. |
| An abstract class can extend another java class and implement multiple java interfaces | An interface can extend another java interface only |
| An abstract class can be extended using keyword “extends” | An interface can be implemented using keyword “implements” |
| A java abstract class can have class members like private, protected etc | Members of a java interface are public by default. |
| Eg:  public abstract class shape  {  public abstract void draw();  } | Eg:  public interface Drawable  {  Void draw();  } |

1. **Difference between Is-a and Has –a?**

|  |  |
| --- | --- |
| **IS-A relationship** | **HAS-A relationship** |
| It is also known as inheritance, it is  Acquiring parent class properties. | It is acquiring class properties by creating instance of that class. |
| It is achieved by “extends” keyword. | It is achieved by “new” keyword. |
| For example, an apple is a fruit. So you will extend fruit to get apple. | For example, a room has a table. So you will create a class room and then in that class create an instance of type table. |
| Inheritance is a static (compile time) binding. | A HAS-A relationship is dynamic (run time) binding |

1. **What is Static?**

* Static is keyword.
* We can apply static keyword with variables, methods, blocks and classes.
* The static variable gets memory only once in the class area at the time of class loading.
* It can be used to refer to the common properties of all objects, for example, the company name of employees, college name of students, etc.
* A static method belongs to the class rather than the object of a class. It can be invoked without the need for creating an instance of a class.
* Static block is used to initialize the static data member. It is executed before the main method at the time of class loading. It can access static data member and can change the value of it. It can be used to setup database connection.

1. **What is non-static block?**

* It is used for non-initializing content.
* Before calling constructor non-static block is executed.

1. **What is Object?**

* An object is an instance of a class.
* Objects have states and behaviours. Example: A dog has states - color, name, breed as well as behaviours – eat, bark, smell.

**Exception:**

1. **What is Exception?**

* In our application, there are chances of abnormal condition, normal flow of program get disturbed, such situation is called as an exception.
* All types of exceptions only occurs at runtime.
* So avoiding such abnormal termination we need to handle exception.
* Exception will occur because of wrong user input.

1. **What is Exception Handling?**

* To avoid occurring exception we need to provide alternative way to execute rest of program, this is called as exception handling.
* Keywords to handle exception: try, catch, finally, throw, throws.

1. **Difference between Exception and Error?**

|  |  |
| --- | --- |
| Exception | Error |
| Classified as checked and unchecked type. | Classified as an unchecked type. |
| It belongs to java.lang.Exception. | It belongs to java.lang.error. |
| We need to handle the exceptions. | We need to solve the errors. |
| It can occur at run-time & compile-time both. | It doesn't occur at compile-time, only occurs at run-time. |
| E.g. NullPointerException , SqlException, etc. | E.g. OutOfMemoryError , AbstractMethodError, etc. |

1. **Default Exception handling in Java?**

* Step-1: Problem Analysed.
* Step-2: Problem Finding.
* Step-3: Create Object of an Exception found.

E.g. ArithmeticException e=new ArithmeticException ( );

* Step-4: Throw e ;
* Step-5: JVM will catch.
* Step-6: JVM will display Exception message.

1. **Difference between Checked Exception & Unchecked Exception?**

|  |  |
| --- | --- |
| Checked Exception | Unchecked Exception |
| Exceptions that compiler forces user to write handling code before compilation, it is called checked exception. | Exceptions that compiler doesn’t forces user to write handling code are called checked exception. |
| Compulsory need to handle them. | Not compulsion to handle them. |
| It increases code. | It reduces code. |
| E.g. IOexception, SqlException, FileNotFoundException, etc. | All Runtime Exception as well as all Errors are example of unchecked exception. |

1. **What is use of finally block?**

* “finally” block code always be executed either there is problem inside “try” block or there is no problem inside “try” block. In both situation, “finally” block code will be executed.
* Resources which are open inside “try” block, those resources should be closed inside “finally” block. E.g. If database connection open then it should be closed inside “finally” block.
* Syntax :

try { try {

} }

finally { catch ( ) {

} }

finally {

}

* We cannot write two or more “finally” block for one “try” block, only one “finally” block can be written with “try” block.
* We cannot write “finally” block before “catch” block, we can write it only after “catch” block like try-catch-finally.

1. **How to stop executing finally block?**

* Before finally block get executed, System.exit(0); line get executed.
* Before finally block get executed, unending loop get executed.

1. **What is final, finally, finalized?**

* The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be used for variables, methods, class.
* Final variable once assigned can’t be changed after.
* Final method can’t be rewritten, can’t be inherited.
* Final class can’t be accessed by creating child of it.
* The finally is a block that always be executed either there is exception occur inside “try” block or there is no exception occur inside “try” block. In both situation, “finally” block code will be executed.
* The finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object, it is used to perform clean-up activity.

1. **What is throws keyword?**

* By using “throws” keyword we can give a chance to caller method to handle the exception.
* The “throws” keyword is used for propagating the exception.
* Whenever unchecked exception will occur, it will automatically propagated.
* Whenever checked exception will occur, we need to write “throws” keyword for propagating the exception.
* e.g. public void m1( ) throws IO Exception, SQL Exception

{ }

1. **What s throw keyword?**

* The “throw” keyword is used for occurring our own exception.
* To occur custom exception we use “throw” keyword.
* Syntax :

public void m1( )

{

Arithmetic Exception e=new Arithmetic Exception( ) ;

throw e;}

OR

public void m2( )

{

throw new Arithmetic Exception( ) ;

}

1. **Difference between throw & throws?**

|  |  |
| --- | --- |
| throw | throws |
| It is used to occur custom exception. | It is used to give a chance to caller method to handle the exception. |
| This keyword is used while creating new instance of exception class. | This keyword is used with method name to declare the exception that need to propagate. |
| Can be used only with single exception at a time. | Can be used with multiple exceptions at a time. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions |

1. **Why to create Customise Exception?**

Customise Exception tells user user-friendly message, as user can’t understand programming language code for exception.

1. **Tell me Customise Exception for any Checked Exception?**

* First create user-define class for customise exception, extend it to corresponding checked exception class, and also create parameterised constructor that accepts string input and forward that variable using super keyword.
* Now use throw keyword and create new instance of custom class you have created where you want to handle checked exception in your application.

1. **Can we create Customise Exception for Checked Exception?**

* Yes we can but not necessary, because customise exception used for telling user user-friendly message as user doesn’t know programming language.
* But programmer is aware of programming codes, so programmer doesn’t require customise exception.

1. **New Exception related feature in JDK 1.7 version?**

* Try with Resource.

1. **Which type of statements can be written in try with resources?**

* We can write those classes/interfaces which have implemented/extended Autoclosable interface only.
* For user-defined class/interface, we need to implement/extend Autoclosable interface explicitly. So that we can write that class in try with resources context.

1. **Difference between ClassCasteException and NoClassDefFound Error?**

|  |  |
| --- | --- |
| ClassCasteException | NoClassDefFound Error |
| It occurs when we are trying to type cast parent object to Child. | It occurs when it is unable to find required .class file at runtime. |
| It can be avoided by correcting type while type casting, by changing in code. | It can be avoided by providing required .class file. |
| It is child class of Runtime Exception. | It is child class Error. |

1. **Different Scenarios in Exception Handling?**

 > Once problem will occur inside “try” block, then remaining line of code of “try” block will not be executed.

* If there is no problem inside “try” block, then “catch” block will be skipped.

In “catch” block we can write same class of exception and its parent classes only.

* We can write two or more “catch” for one “try” block, but we can write child classes of exception in first catch blocks and then parent class exception, we cannot write first parent exception before child exception.

 > If “try” with “finally” block and “try” have return statement, then before control return back to caller first of all “finally” block code will be executed.

* If “try” with “finally” block, “try” have return statement and if “finally” block change return statement value, then it will be changed only for “finally” block, it will not be changed for caller.
* If “try” and “finally” both have return statement, then “finally” block statement will be executed.

 “throws” with method calling, in case of unchecked exception classes :

* For unchecked exceptions, it is not needed to write explicitly throws exception, the exception propagation is done implicitly to the caller.
* In case of checked exception classes, needed to write explicitly throws exception for propagating exception to the caller.

 “throws” keyword with method Overriding:

* (In case of unchecked exceptions) If Parent class method throws unchecked exception or doesn’t throws any exception then at the time of overriding no need to write throws keyword, if we want we can write ay of unchecked exception class but not checked exception class.
* (In case of checked exceptions) If Parent class method throws checked exception then at the time of overriding no need to write throws keyword, if we want then we can write same exception class or their child class but not parent class.

 “throws” keyword with Super Class constructor and Sub Class constructor:

* (For unchecked exception) If Parent class constructor doesn’t throws any exception or “throws” unchecked exception then child class constructor no need to write “throws keyword, if we want then any of exception class they can write.
* (For checked exception) If parent class constructor throws exception then at a time of inheritance child class constructor must write “throws” keyword with same exception class or their parent class.

1. **How do you handled Exception in your project?**

* We created separate package for Exception classes needed in project where handling code was written, then at a time of exception handling scenario in project used throws keyword.

**Collection:**

1. **What is Collection?**

* The Collection is a framework that provides an architecture to store and manipulate the group of objects.
* t provides operations that you perform on a data such as searching, sorting, insertion and deletion on the group of objects.
* Collection represents a single unit of objects as a group.

1. (Explain Difference between Array & Collection)

|  |  |
| --- | --- |
| Array | Collection |
| It is Fixed in Size. | It is Growable in nature. |
| It can hold only Homogeneous Data  Elements. | It can hold both Homogeneous and  Heterogeneous Elements. |
| With Respect to Memory Arrays are Not  Recommended to Use. | With Respect to Memory Collections are  Recommended to Use. |

1. **What is Difference between Collection & Collections?**

|  |  |
| --- | --- |
| Collection | Collections |
| It is an interface. | It is class. |
| It can be used to Represent a Group of Individual Objects as a Single Entity. | It is used to sort and synchronize the collection elements. |
| It provides the methods that can be used for data structure. | It provides the methods which can be used for various operations on a collection. |

1. **How Arraylist works?**

* When we create object of Arraylist, it create Arraylist instance with default capacity 10.
* Arraylist capacity increases with formula – New Capacity = ((3/2) x Old Capacity)+1
* When Arraylist increments with new capacity then data from old Arraylist is copied into new instance and old instance is destroyed.
* When we add or delete data into the Arraylist then multiple data shift operations are performed.
* Arraylist follows Indexing.

1. **Difference between Arraylist & Vector?**

|  |  |
| --- | --- |
| Arraylist | Vector |
| It is not Synchronised. | It is Synchronised. |
| It is not Thread Safe | It is Thread Safe |
| Its Default Capacity :- 10 and It increases after by (Old+Old\*(3/2))+1 | Its Default Capacity :- 10 and It increases after by Old+Old\*100% |
| Performance is high. | Performance is low. |
| Enumeration cannot be used. | Enumeration can be used. |

1. **Why Arraylist is fast for retrieval operation?**

* It implements RandomAccess Interface, hence Arraylist is fast for retrieval operation.

1. **Why set doesn’t allow duplicates?**

* Set internally uses HashMap.
* HashMap object is created in every Set implemented class.
* Here HashMap stores key as all inserted elements and value as a dummy object created with new keyword.
* As HashMap doesn’t accept duplicate keys, so set don’t allow duplicate values.

1. **How Linkedlist works? (Why insertion & deletion is fast in Linkedlist?)**

* When we create an object of Linkedlist and add an element to it.
* It stores element as a node in which previous & next node address is also stored.
* Node format = ||prev. node addr.| (value) |next node addr.||
* Due to previous & next node address is stored, hence while updation or insertion & deletion operation data shift operation need not to perform and it makes Linkedlist fast.

1. **Define Linkedlist?**

* It is one of implemented class of List interface in collection framework.
* It allows duplicate values, Insertion order is preserved & indexing is maintained.
* It implements Clonable, Serializable interfaces.
* It follows doubly linked list structure.
* It is mostly preferable for insertion & deletion operations.

1. **What is Map?**

* Map is used for store different object in the pair of “key” and “value”.
* In map, “key” should be unique.
* Insertion order will not be maintained in Map.

1. **Difference between Hashmap & Hashtable?**

|  |  |
| --- | --- |
| HashMap | Hashtable |
| It is not Synchronised. | It is Synchronised. |
| It allows multiple threads at a time. | It allows single thread at a time. |
| It is not thread safe. | It is thread safe. |
| Null key (once) & Null value is allowed. | Null key & Null value is not allowed. |
| Its performance is fast. | Its performance is slow. |

1. **How HashMap works?**

* When we create HashMap object, HashMap instance as per default capacity 16 buckets is created.
* When we perform add (put ( )) operation, it accepts data in key & value format.
* Internally hashing technique is used, that generates hashcode for key and also calculate index to find bucket location for inserting data in HashMap instance.
* It will store element at that location as a node format.

||previous node address| (Key) | (Value) |next node address||

* Now when we perform retrieval (get ( )) operation, it asks for key.
* Again hashing technique is used and bucket location is identified, then equals ( ) method is used to compare key content and if it returns true then value is retrieved.

1. **What happens when we put same keys in Map?**

* If we add a key-value pair where the key exists already, put method replaces the existing value of the key with the new value.

1. **What is Contract between equals () & hashcode ()?**

* If equals ( ) returns true, then objects must have same hashcodes.
* If equals ( ) returns false, then objects may or may not have same hashcodes.
* If hashcodes of objects are same, then we can’t conclude output of equals ( ), it may be true or may be false.
* If hashcodes of objects are different, then output of equals ( ) must be false.

1. **Difference between Hashmap & Synchronised (or Concurrent) Hashmap?**

|  |  |
| --- | --- |
| HashMap | Synchronised or Concurrent HashMap |
| It is non-Synchronized in nature. | It is Synchronized in nature. |
| It is not Thread-safe. | It is thread-safe. |
| Performance is high. | Performance is low. |
| It can throw ConcurrentModificationException. | It doesn’t throw ConcurrentModificationException |

1. **Difference between Comparable & Comparator?**

|  |  |
| --- | --- |
| Comparable | Comparator |
| This interface is from java.lang package. | This interface is from java.util package. |
| It is used for Default sorting. | It is used for Custom sorting. |
| It has only one method i.e. compareTo. | It has two methods i.e. compare & equals. |
| Programmer decides how sorting is to be done. | User decides how sorting is to be done. |

1. **What is fail safe & fail fast iterator?**

* Using iterators we can traverse over the collections objects. The iterators can be either fail-safe or fail-fast.
* Fail-safe iterator means they will not throw any exception even if the collection is modified while iterating over it.
* Whereas Fail-fast iterator throw an exception (ConcurrentModificationException) if the collection is modified while iterating over it.
* Fail-Fast Iterators internal working:

o Every fail fast collection has a modCount field, to represent how many times the collection has changed/modified.

o So at every modification of this collection we increment the modCount value. For example the modCount is incremented in below cases:

1. When one or more elements are removed.

2. When one or more elements are added.

3. When the collection is replaced with other collection.

4. When the collection is sorted

o So everytime there is some change in the collection structure, the mod count is incremented.

o Now the iterator stores the modCount value in the initialization as below:

int expectedModCount = modCount;

o Now while the iteration is going on, expectedModCount will have old value of modCount. If there is any change made in the collection, the modCount will change and then an exception is thrown

* Unlike the fail-fast iterators, fail-safe iterators traverse over the clone of the collection. So even if the original collection gets structurally modified, no exception will be thrown.

1. **What is WeakHashmap?**

* In Case of WeakHashMap if an Object doesn't contain any References then it is Always Eligible for GC Even though it is associated with WeakHashMap.
* Garbage Collector Dominates WeakHashMap.
* Both null values and null keys are supported in WeakHashMap.
* It is not synchronised.

**String:**

1. **How to declare a string in Java?**

* String declaration in Java can be done in two ways:
* **By string literal**: Double quotes are used to create Java String literals.
* Example: String str= "Scaler";
* **By new keyword**: Keyword "new" is used to create a Java string.
* Example: String str=new String ("Scaler");

1. **Is String a primitive or derived type in Java?**

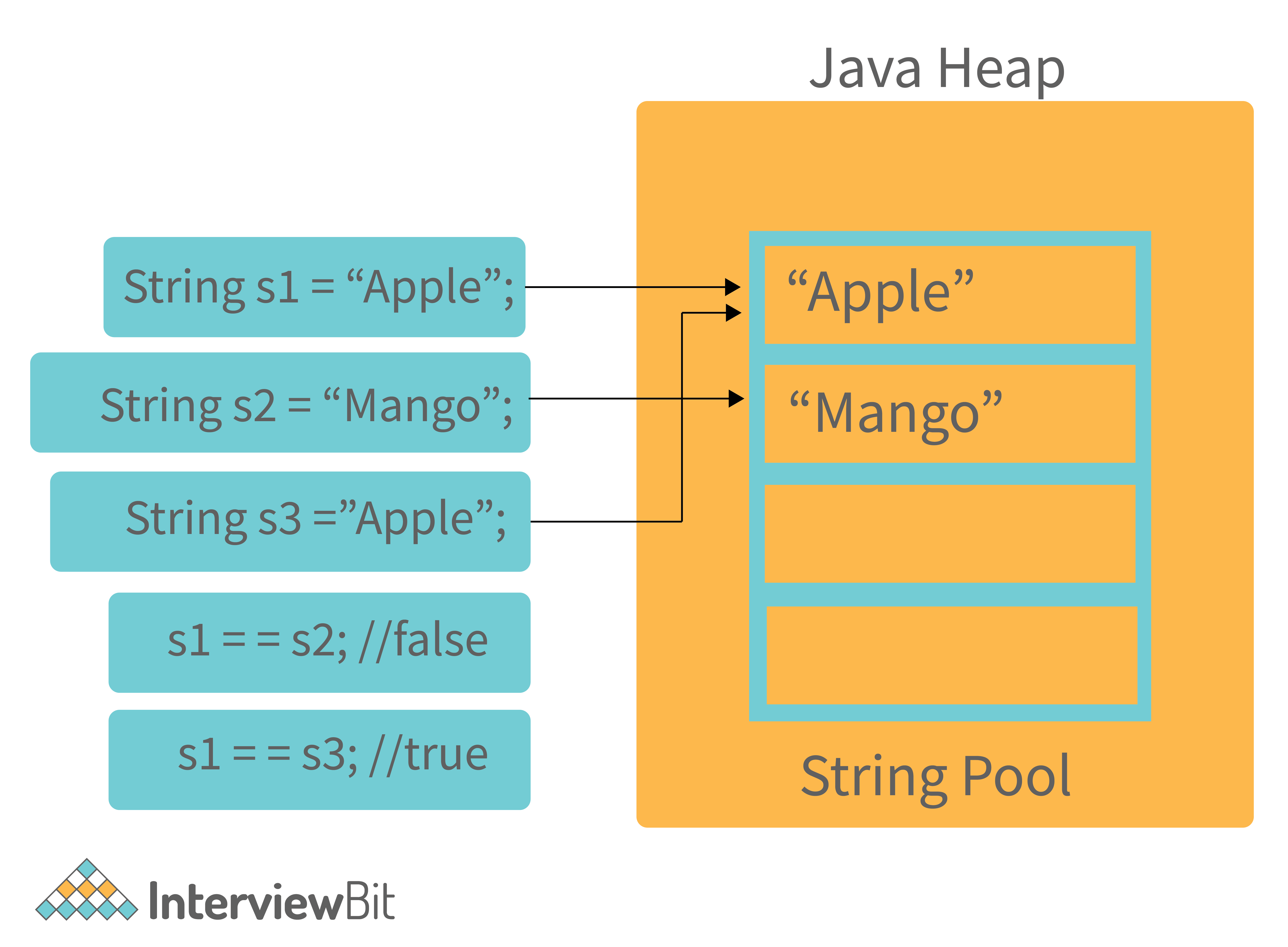
* Strings are derived data types. Strings are Java objects that represent sequences of characters. String objects are created using the java.lang.String class. There are many functions that need to be called upon when processing a string, such as substring(), indexof(), equals(), toUppercase(), etc, which primitives types do not have.

1. **State the difference between String in C and String in Java.**

* **String in C**: In C, strings are just arrays of characters, and they are terminated with a /0, which is why we commonly refer to them as "null-terminated". Strings in C, like "abc$%", actually consist of 6 characters 'a' 'b' 'c' '$' '%' and '/0', but these can be easily manipulated.
* **String in Java**: Java treats Strings as objects, not arrays. String objects are created using the java.lang.String class. String objects in Java are immutable; you cannot modify their contents. This means whenever we manipulate a String object, the new String is created rather than the original string being modified.

### Explain String pool in java?

* String Pool, also known as SCP (String Constant Pool), is a special storage space in Java heap memory that is used to store unique string objects. Whenever a string object is created, it first checks whether the String object with the same string value is already present in the String pool or not, and if it is available, then the reference to the string object from the string pool is returned. Otherwise, the new string object is added to the string pool, and the respective reference will be returned.



* As shown in the above image, two Strings s1 and s2 are created with the values "Apple" and "Mango". Therefore, when the third String s3 containing the value "Apple" is created, instead of creating a new object, the existing object reference will be returned. Here, s1==s2 is false both strings s1 and s2 refer to different string values from the string pool i.e. apple and mango. We can see that s1==s3 is true because both strings s1 and s3 refer to a single string value from a string pool i.e., apple.

1. **Is String immutable or final in Java? If so, then what are the benefits of Strings being Immutable?**

* Yes, Strings are immutable in Java. Immutable objects mean they can't be changed or altered once they've been created. However, we can only modify the reference to the string object. The String is immutable in Java because of many reasons like security, caching, synchronization and concurrency, and class loading.

1. **What does the string intern() method do in Java?**

* If you apply the intern() method to a few strings, you will ensure that all strings having the same content share the same memory. As soon as a String object is invoked with intern(), it first checks if the string value of the String object is already present in the string pool and if it is available, then the reference to that string from the string constant pool is returned. If not, a new string object is added to the string pool, and a reference to it is returned.
* Eg:

String str1 = new String("Scaler by InterviewBit").intern(); //Line1

String str2 = new String("Scaler by InterviewBit").intern(); //Line2

System.out.println(str1 == str2); //prints true

* As you can see, the intern() method is invoked on the String objects. When Line1 is executed, memory is allocated within the SCP. In line 2, no new string objects are created in the SCP because str1 and str2 have the same content. As a result, the reference to the object created in line1 is returned. This means that str1 and str2 both point to the same memory. Therefore, the print statement prints true.

1. **State the difference between String and StringBuffer.**

* String objects in Java are immutable and final, so we can't change their value after they are created. Since strings are commonly used in applications, we need to perform several operations on them such as substring(), equals(), indexof(), toUppercase(), etc. Each time we manipulate a string, a new String object is created, and all previous objects will be garbage, placing a strain on the garbage collector. This is why The Java team developed StringBuffer. A StringBuffer is a mutable object, meaning it can be changed, but the string is an immutable object, so it cannot be changed once it has been created.
* **String**

**Syntax:**

String str1="InterviewBit";

String str2=new String("Scaler");

Scanner str3=new Scanner(System.in);

String str4=str3.nextLine();

**Example:**Concatenation Example of String. A string class takes longer to perform a concatenation operation than a string buffer class.

**public** **class** **Scanner**

{

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

{

StringBuilder stbu=**new** StringBuilder();

//Initial object size

System.out.println(stbu.capacity());

String str="Scaler";

System.out.println(str);

String str1 = **new** String("InterviewBit");

System.out.println(str1);

str1 += " Articles"; //string update

System.out.println(str1);

}

}

**Output:**

16

Scaler

InterviewBit

InterviewBit Articles

* **StringBuffer**

**Syntax:**

StringBuffer var = new StringBuffer(str);

**Example:** Concatenation Example of StringBuffer. String buffer class perform concatenation operations more quickly than string classes.

**public** **class** **StringBuffer**

{

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

{

StringBuilder stbu=**new** StringBuilder();

//Initial object size

System.out.println(stbu.capacity());

StringBuffer stbr= **new** StringBuffer("InterviewBit");

System.out.println(stbr);

stbr.append(" Articles"); //string update

System.out.println(stbr);

stbr=**new** StringBuffer("Scaler");

System.out.println(stbr);

}

}

Output:

16

InterviewBit

InterviewBit Articles

Scaler

1. **State the difference between StringBuffer and StringBuilder in Java.**

* StringBuffer and StringBuilder are two Java classes for manipulating strings. These are mutable objects, i.e., they can be modified, and provide various methods such as insert(), substring(), delete(), and append(), for String manipulation.
* **StringBuffer**: The StringBuffer class was created by the Java Team when they realized the need for an editable string object. Nevertheless, StringBuffer has all methods synchronized, meaning they are thread-safe. Therefore, StringBuffer allows only one thread to access a method at once, so it is not possible to call StringBuffer methods from two threads simultaneously, which means it takes more time to access. The StringBuffer class has synchronized methods, making it thread-safe, slower, and less efficient than StringBuilder. The StringBuffer class was introduced in Java 1.0.
* Syntax:

StringBuffer var = new StringBuffer(str);

* **StringBuilder**: It was at that point that the Java Team realized that making all methods of StringBuffer synchronized wasn't the best idea, which led them to introduce StringBuilder. The StringBuilder class has no synchronized methods. Unlike StringBuffer, StringBuilder does not offer synchronized methods, which makes it less thread-safe, faster, and more efficient. StringBuilder was introduced in Java 1.5 in response to StringBuffer's shortcomings.

Syntax:

StringBuilder var = new StringBuilder(str);

1. **In Java, how can two strings be compared?**

In Java, there are several ways for comparing two strings. The following are a few of them:

* **String Equals Method**: In this method, the strings are compared based on the values within them. If the values of the two strings are the same, it returns true; otherwise, it returns false. This method is case-sensitive.

Syntax:

str1.equals(str2);

For example:

Input 1= Scaler

Input 2= InterviewBit

Output= **false**

Input 1= Scaler

Input 2= Scaler

Output= **true**

Input 1= Scaler

Input 2= scaler

Output= **false**

* **String Equals Ignore Case**: By using this method, the two strings are compared without taking into account the case (upper or lower). It returns true if the two values are the same and not null.  
    
  Syntax:

str1.equalsIgnoreCase(str2);

For Example:

Input 1= Scaler

Input 2= InterviewBit

Output= **false**

Input 1= Scaler

Input 2= Scaler

Output= **true**

Input 1= Scaler

Input 2= scaler

Output= **true**

* **Object Equals Method**: The method returns true if its arguments are equal, otherwise, it returns false. Accordingly, if both arguments are null, the result is true, and if just one argument is null, the result is false.
* Syntax:

Object.equals(str1, str2)

For example:

Input 1= Scaler

Input 2= InterviewBit

Output= **false**

Input 1= Scaler

Input 2= Scaler

Output= **true**

Input 1= Scaler

Input 2= **null**

Output= **false**

Input 1= **null**

Input 2= **null**

Output= True

* **String Compare To Method**: This method compares input strings with each other. Upon comparison, the following value is returned:

1. If (str1>str2), a positive value is returned.
2. If (str1==str2), 0 is returned.
3. If (str1<str2), a negative value is returned.

Syntax:

str1.compareTo(str2)

Example:

Input 1= InterviewBit

Input 2= Scaler

Output= -10

Input 1= Scaler

Input 2= Scaler

Output= 0

Input 1= Scaler

Input 2= InterviewBit

Output= 10

**10.What is the difference between str1 == str2 and str1.equals(str2)?**

Java offers both the equals() method and the "==" operator for comparing objects. However, here are some differences between the two:

* Essentially, equals() is a method, while == is an operator.
* The == operator can be used for comparing references (addresses) and the .equals() method can be used to compare content. To put it simply, == checks if the objects point to the same memory location, whereas .equals() compares the values of the objects.

*Example:*

**public** **class** **StringComparison**

{

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

{

String str1=**new** String("Scaler");

String str2=**new** String("Scaler");

System.out.println(str1 == str2);

System.out.println(str1.equals(str2));

}

}

Output:

false

true

In this example, two different String objects are being created, str1 and str2.

* If str1 and str2 are compared using the == operator, then the result will be false, because both have different addresses in the memory. Both must have the same address in the memory for the result to be true.
* If you use the equals method, the result is true since it's only comparing the values given to str1 and str2, even though they are different objects.

**JAVA 8:**

1. **Describe the newly added features in Java 8?**

Here are the newly added features of Java 8:

| Feature Name | Description |
| --- | --- |
| Lambda expression | A function that can be shared or referred to as an object. |
| Functional Interfaces | Single abstract method interface. |
| Method References | Uses function as a parameter to invoke a method. |
| Default method | It provides an implementation of methods within interfaces enabling 'Interface evolution' facilities. |
| Stream API | Abstract layer that provides pipeline processing of the data. |
| Date Time API | New improved joda-time inspired APIs to overcome the drawbacks in previous versions |
| Optional | Wrapper class to check the null values and helps in further processing based on the value. |
| Nashorn, JavaScript Engine | An improvised version of JavaScript Engine that enables JavaScript executions in Java, to replace Rhino. |

1. **In which programming paradigm Java 8 falls?**

* Object-oriented programming language.
* Functional programming language.
* Procedural programming language.
* Logic programming language
* 3. What are the significant advantages of Java 8?
* Compact, readable, and reusable code.
* Less boilerplate code.
* Parallel operations and execution.
* Can be ported across operating systems.
* High stability.
* Stable environment.
* Adequate support

1. **What is MetaSpace? How does it differ from PermGen**

JVM

* PremGen: MetaData information of classes was stored in PremGen (Permanent-Generation) memory type before Java 8. PremGen is fixed in size and cannot be dynamically resized. It was a contiguous Java Heap Memory.
* MetaSpace: Java 8 stores the MetaData of classes in native memory called 'MetaSpace'. It is not a contiguous Heap Memory and hence can be grown dynamically which helps to overcome the size constraints. This improves the garbage collection, auto-tuning, and de-allocation of metadata.

1. **What are functional or SAM interfaces?**

* Functional Interfaces are an interface with only one abstract method. Due to which it is also known as the Single Abstract Method (SAM) interface. It is known as a functional interface because it wraps a function as an interface or in other words a function is represented by a single abstract method of the interface.
* Functional interfaces can have any number of default, static, and overridden methods. For declaring Functional Interfaces @FunctionalInterface annotation is optional to use. If this annotation is used for interfaces with more than one abstract method, it will generate a compiler error.

@FunctionalInterface // Annotation is optional

public interface Foo() {

// Default Method - Optional can be 0 or more

public default String HelloWorld() {

return "Hello World";

}

// Static Method - Optional can be 0 or more

public static String CustomMessage(String msg) {

return msg;

}

// Single Abstract Method

public void bar();

}

public class FooImplementation implements Foo {

// Default Method - Optional to Override

@Override

public default String HelloWorld() {

return "Hello Java 8";

}

// Method Override

@Override

public void bar() {

System.out.println(“Hello World”);

}

}

public static void main(String[] args) {

FooImplementation fi = new FooImplementation();

System.out.println(fi.HelloWorld());

System.out.println(fi.CustomMessage(“Hi”));

fi.bar();

}

1. **Can a functional interface extend/inherit another interface?**

* A functional interface cannot extend another interface with abstract methods as it will void the rule of one abstract method per functional interface. E.g:

interface Parent {

public int parentMethod();

}

@FunctionalInterface // This cannot be FunctionalInterface

interface Child extends Parent {

public int childMethod();

// It will also extend the abstract method of the Parent Interface

// Hence it will have more than one abstract method

// And will give a compiler error

}

It can extend other interfaces which do not have any abstract method and only have the default, static, another class is overridden, and normal methods. For eg:

interface Parent {

public void parentMethod(){

System.out.println("Hello");

}

}

@FunctionalInterface

interface Child extends Parent {

public int childMethod();

}

1. **What is the default method, and why is it required?**

* A method in the interface that has a predefined body is known as the default method. It uses the keyword default. default methods were introduced in Java 8 to have 'Backward Compatibility in case JDK modifies any interfaces. In case a new abstract method is added to the interface, all classes implementing the interface will break and will have to implement the new method. With default methods, there will not be any impact on the interface implementing classes. default methods can be overridden if needed in the implementation. Also, it does not qualify as synchronized or final.

@FunctionalInterface // Annotation is optional

public interface Foo() {

// Default Method - Optional can be 0 or more

public default String HelloWorld() {

return "Hello World";

}

// Single Abstract Method

public void bar();

}

1. **What are static methods in Interfaces?**

* Static methods, which contains method implementation is owned by the interface and is invoked using the name of the interface, it is suitable for defining the utility methods and cannot be overridden.

1. **What are some standard Java pre-defined functional interfaces?**

* Some of the famous pre-defined functional interfaces from previous Java versions are Runnable, Callable, Comparator, and Comparable. While Java 8 introduces functional interfaces like Supplier, Consumer, Predicate, etc. Please refer to the java.util.function doc for other predefined functional interfaces and its description introduced in Java 8.
* Runnable: use to execute the instances of a class over another thread with no arguments and no return value.
* Callable: use to execute the instances of a class over another thread with no arguments and it either returns a value or throws an exception.
* Comparator: use to sort different objects in a user-defined order
* Comparable: use to sort objects in the natural sort order

1. **What are the various categories of pre-defined function interfaces?**

* Function: To transform arguments in returnable value.
* Predicate: To perform a test and return a Boolean value.
* Consumer: Accept arguments but do not return any values.
* Supplier: Do not accept any arguments but return a value.
* Operator: Perform a reduction type operation that accepts the same input types.

1. **What is the lambda expression in Java and How does a lambda expression relate to a functional interface?**

* Lambda expression is a type of function without a name. It may or may not have results and parameters. It is known as an anonymous function as it does not have type information by itself. It is executed on-demand. It is beneficial in iterating, filtering, and extracting data from a collection.
* As lambda expressions are similar to anonymous functions, they can only be applied to the single abstract method of Functional Interface. It will infer the return type, type, and several arguments from the signature of the abstract method of functional interface.

1. **What is the basic structure/syntax of a lambda expression?**

FunctionalInterface fi = (String name) -> {

System.out.println("Hello "+name);

return "Hello "+name;

}

1. **Lambda expression can be divided into three distinct parts as below:**

1. List of Arguments/Params:

(String name)

A list of params is passed in () round brackets. It can have zero or more params. Declaring the type of parameter is optional and can be inferred for the context.

2. Arrow Token:

->   
Arrow token is known as the lambda arrow operator. It is used to separate the parameters from the body, or it points the list of arguments to the body. 3. Expression/Body:

{

System.out.println("Hello "+name);

return "Hello "+name;

}

A body can have expressions or statements. {} curly braces are only required when there is more than one line. In one statement, the return type is the same as the return type of the statement. In other cases, the return type is either inferred by the return keyword or void if nothing is returned.

1. **What are the features of a lambda expression?**

Below are the two significant features of the methods that are defined as the lambda expressions:

Lambda expressions can be passed as a parameter to another method.

Lambda expressions can be standalone without belonging to any class.

1. **What is a type interface?**

* Type interface is available even in earlier versions of Java. It is used to infer the type of argument by the compiler at the compile time by looking at method invocation and corresponding declaration.

1. **What are the types and common ways to use lambda expressions?**

* A lambda expression does not have any specific type by itself. A lambda expression receives type once it is assigned to a functional interface. That same lambda expression can be assigned to different functional interface types and can have a different type.
* For eg consider expression s -> s.isEmpty() :

Predicate<String> stringPredicate = s -> s.isEmpty();   
Predicate<List> listPredicate = s -> s.isEmpty();  
Function<String, Boolean> func = s -> s.isEmpty();  
Consumer<String> stringConsumer = s -> s.isEmpty();

Common ways to use the expression

Assignment to a functional Interface —> Predicate<String> stringPredicate = s -> s.isEmpty();  
Can be passed as a parameter that has a functional type —> stream.filter(s -> s.isEmpty())  
Returning it from a function —> return s -> s.isEmpty()  
Casting it to a functional type —> (Predicate<String>) s -> s.isEmpty()

1. **In Java 8, what is Method Reference?**

* Method reference is a compact way of referring to a method of functional interface. It is used to refer to a method without invoking it. :: (double colon) is used for describing the method reference. The syntax is class::methodName

For e.g.:

Integer::parseInt(str) \\ method reference

str -> Integer.ParseInt(str); \\ equivalent lambda

1. **What does the String::ValueOf expression mean?**

* It is a static method reference to method Valueof() of class String. It will return the string representation of the argument passed.

1. **What is an Optional class?**

* Optional is a container type which may or may not contain value i.e. zero(null) or one(not-null) value. It is part of java.util package. There are pre-defined methods like isPresent(), which returns true if the value is present or else false and the method get(), which will return the value if it is present.

static Optional<String> changeCase(String word) {

if (name != null && word.startsWith("A")) {

return Optional.of(word.toUpperCase());

}

else {

return Optional.ofNullable(word); // someString can be null

}

}

1. **What are the advantages of using the Optional class?**

* Below are the main advantage of using the Optional class:

It encapsulates optional values, i.e., null or not-null values, which helps in avoiding null checks, which results in better, readable, and robust code It acts as a wrapper around the object and returns an object instead of a value, which can be used to avoid run-time NullPointerExceptions.

1. **What are Java 8 streams?**

* A stream is an abstraction to express data processing queries in a declarative way.
* A Stream, which represents a sequence of data objects & series of operations on that data is a data pipeline that is not related to Java I/O Streams does not hold any data permanently.  
  The key interface is java.util.stream.Stream<T>. It accepts Functional Interfaces so that lambdas can be passed. Streams support a fluent interface or chaining. Below is the basic stream timeline marble

1. **What are the main components of a Stream?**

Components of the stream are:

* A data source
* Set of Intermediate Operations to process the data source
* Single Terminal Operation that produces the result
* Components of Stream

1. **What are the sources of data objects a Stream can process?**

* A Stream can process the following data:
* A collection of an Array.
* An I/O channel or an input device.
* A reactive source (e.g., comments in social media or tweets/re-tweets)
* A stream generator function or a static factory.

1. **What are Intermediate and Terminal operations?**

**Intermediate Operations:**

* Process the stream elements.
* Typically transforms a stream into another stream.
* Are lazy, i.e., not executed till a terminal operation is invoked.
* Does internal iteration of all source elements.
* Any number of operations can be chained in the processing pipeline.
* Operations are applied as per the defined order.
* Intermediate operations are mostly lambda functions.

**Terminal Operations:**

* Kick-starts the Stream pipeline.
* used to collect the processed Stream data.
* int count = Stream.of(1, 2, 3, 4, 5)
* .filter(i -> i <4) // Intermediate Operation filter
* .count(); // Terminal Operation count

1. **What are the most commonly used Intermediate operations?**

* Filter(Predicate<T>) - Allows selective processing of Stream elements. It returns elements that are satisfying the supplied condition by the predicate.
* map(Funtion<T, R>) - Returns a new Stream, transforming each of the elements by applying the supplied mapper function.= sorted() - Sorts the input elements and then passes them to the next stage.
* distinct() - Only pass on elements to the next stage, not passed yet.
* limit(long maxsize) - Limit the stream size to maxsize.
* skip(long start) - Skip the initial elements till the start.
* peek(Consumer) - Apply a consumer without modification to the stream.
* flatMap(mapper) - Transform each element to a stream of its constituent elements and flatten all the streams into a single stream.

1. **What is the stateful intermediate operation? Give some examples of stateful intermediate operations.**

* To complete some of the intermediate operations, some state is to be maintained, and such intermediate operations are called stateful intermediate operations. Parallel execution of these types of operations is complex.
* For Eg: sorted() , distinct() , limit() , skip() etc.
* Sending data elements to further steps in the pipeline stops till all the data is sorted for sorted() and stream data elements are stored in temporary data structures.

1. **What is the most common type of Terminal operations?**

collect() - Collects single result from all elements of the stream sequence.

reduce() - Produces a single result from all elements of the stream sequence

count() - Returns the number of elements on the stream.

min() - Returns the min element from the stream.

max() - Returns the max element from the stream.

Search/Query operations

anyMatch() , noneMatch() , allMatch() , ... - Short-circuiting operations.

Takes a Predicate as input for the match condition.

Stream processing will be stopped, as and when the result can be determined.

Iterative operations

forEach() - Useful to do something with each of the Stream elements. It accepts a consumer.

forEachOrdered() - It is helpful to maintain order in parallel streams.

1. **What is the difference between findFirst() and findAny()?**

| findFirst() | findAny() |
| --- | --- |
| Returns the first element in the Stream | Return any element from the Stream |
| Deterministic in nature | Non-deterministic in nature |

1. **How are Collections different from Stream?**

* Collections are the source for the Stream. Java 8 collection API is enhanced with the default methods returning Stream<T> from the collections.

| Collections | Streams |
| --- | --- |
| Data structure holds all the data elements | No data is stored. Have the capacity to process an infinite number of elements on demand |
| External Iteration | Internal Iteration |
| Can be processed any number of times | Traversed only once |
| Elements are easy to access | No direct way of accessing specific elements |
| Is a data store | Is an API to process the data |

1. **What is the feature of the new Date and Time API in Java 8?**

* Immutable classes and Thread-safe
* Timezone support
* Fluent methods for object creation and arithmetic
* Addresses I18N issue for earlier APIs
* Influenced by popular joda-time package
* All packages are based on the ISO-8601 calendar system

1. **What are the important packages for the new Data and Time API?**

* java.time
* dates
* times
* Instants
* durations
* time-zones
* periods
* Java.time.format
* Java.time.temporal
* java.time.zone

1. **Explain with example, LocalDate, LocalTime, and LocalDateTime APIs.**

**LocalDate**

* Date with no time component

Default format - yyyy-MM-dd (2020-02-20)

LocalDate today = LocalDate.now();  // gives today’s date

LocalDate aDate = LocalDate.of(2011, 12, 30); //(year, month, date)

**LocalTime**

Time with no date with nanosecond precision

Default format - hh:mm:ss:zzz (12:06:03.015) nanosecond is optional

LocalTime now = LocalTime.now();  // gives time now

LocalTime aTime2 = LocalTime.of(18, 20, 30); // (hours, min, sec)

**LocalDateTime**

Holds both Date and Time

Default format - yyyy-MM-dd-HH-mm-ss.zzz (2020-02-20T12:06:03.015)

LocalDateTime timestamp = LocalDateTime.now(); // gives timestamp now

//(year, month, date, hours, min, sec)

LocalDateTime dt1 = LocalDateTime.of(2011, 12, 30, 18, 20, 30);

1. **Define Nashorn in Java 8**

* Nashorn is a JavaScript processing engine that is bundled with Java 8. It provides better compliance with ECMA (European Computer Manufacturers Association) normalized JavaScript specifications and better performance at run-time than older versions.

1. **What is the use of JJS in Java 8?**

* As part of Java 8, JJS is a command-line tool that helps to execute the JavaScript code in the console. Below is the example of CLI commands:

JAVA>jjs  
jjs> print("Hello, Java 8 - I am the new JJS!")  
Hello, Java 8 - I am the new JJS!  
jjs> quit()  
>>