**JAVA 8**

**Java 8 New features:**

(1) Java8 was introduced conciseness in code.

(2) Java brings functional programming which is enabled by lambda expressions (A power full tool to create concise code base)

(3) If you have ever observed, with Python, Scala we can do the same thing in very less Loc (line of code).

By Mid 20s, java lost a large market due to these languages. To prevent further loss java upgraded itself from only

Oops language to some concepts of function programming to create concise code base

**New Features got introduces in java 8:**

(1) Lambda Expressions

(2) Stream API

(3) Default methods in interface.

(4) Static methods

(5) Functional Interface

(6) Optional

(7) Method references

(8) Date API

(9) Nashorn, JavaScript Engine

**Main Advantages of java 8:**

(1) Compact code (Less boilerplate code)

(2) More readable and reusable code

(3) More testable code

(4) Parallel operations.

**(Q) What is lambda expression?**

Lambda expression is an anonymous function (without name, return type and access modifier and having one lambda (->) symbol).

**Normal Programming technique:**

Public void add (int a, int b) {

System.out.print (a+b);

}

**Equivalent Lambda expression:** (a, b) --> System.out.println (a+b);

Lambda expressions are referenced by functional interface.

**(Q) What are functional interfaces?**

**(1) Functional interface are those interfaces can have only one-abstract methods.**

**Reason:**

When you declareda lambda expression and assigning to a variable, it will check the method signature of a method declared inside interface, so java compiler will confuse to which method it needs to create a lambda expression.

(2) It can have any number of static method, default methods. No restrictions on that.

(3)There are many functional interfaces already present in java such as

Ex: Comparable, Runnable

**(Q) How lambda and function interfaces are related?**

Functional interface is used to provide reference to lambda expression.

**Comparator<String> c = (s1, s2) -> s1.compareTo (s2)**

(s1, s2) -> s1.compareTo (s2): This is lambda expression

Comparator<String> c : This is functional interface

**(Q)** **Can you create your own function interface in java?**

As we know, functional interface is an interface with one single abstract method and can have multiple static or default methods.

**To create our own function interface, you can do following steps:**

Create an interface

Annotate that with @Function Interface

Define exactly one Abstract method

There is no restriction on number of static and default methods defined in such interface.

java can implicitly identify functional interface but still you can also annotate it with @FunctionalInterface.It just give the security that incase if u by mistake add 2 abstract methods or remove all abstract methods then compiler will throw compile time error

**(Q) What is method reference in java?**

Method reference is replacement of lambda expressions. It is used to refer method of functional interface to an existing method. Mainly used for code reusability.

Function interfaces abstract method mapped to specific existing method using double colon (::)

This is method reference. Hence, method reference is an alternative to lambda expressions.

Whenever we have existing implementation of abstract method of our functional interface then we go for method reference. If no such method like test Implementation () is available then go for lambda expression.

**(Q) What are default methods?**

Default method is a way of adding new methods to interface without affecting the implementing classes hence; with the new feature java people defended, many compile time errors that may arise due to unimplemented methods of interface.

**Example:**

Interface Animal {

Void legs ();

//default void eyes () {

Sysou("two eyes")

}

}

The use of default method is "Backward Compatibility which means if jdk modifies any interface (Without default method)". Then the classes, which implement this interface, will break.

On the other hand, if you add the default method in interface then you will be able to provide the default implementation.

This will not affect the implementing classes.

**(Q) Is it necessary to override default methods?**

Default methods have dummy implementation

Implementing classes if ok with dummy implementation then the use dummy implementation of default methods. If not satisfied then they can override and provide their own implementation**.**

**(Q) Is Default keyword one of the access modifier?**

(1)Default is not the access modifier like public, protected, or private. (If we dot use any access modifier, then access modifier is default)

(2)For Default access modifier we do not use any keyword.

(3)Hence, Default Keyword was only used in classes until 1.8 version for switch case only but never in interface.

(4)Now it is used for default methods in interface to provide a default implementation for all implementing classes to use.

**(Q) How to override default methods?**

(1) You can override default method by keeping same method signature (name + arguments)

(2) Removing default keywords because in class default keyword is use in switch case to denote default case if no previous case matched. Therefore, you cannot use default keyword in class.

(3) Adding public as Access modifier because in java 8, by default all methods are public so in child u can reduce visibility of overridden default method.

(4) Giving your own implementation.

(**Q) Can you use hash code () default implementation in interface?**

(a)You cannot give your default implementation of hash code () in interface for all implementing classes to use.

(b) We are not allowed to override Object classes’ method as default methods in interface else will get compile time error.

(c) All implementing classes by default has access to all methods of object class

**(Q) How default methods in interface cope up with diamond problem?**

Diamond problem of default methods of interface:

Diamond problem explanation: if two implemented interfaces contains same default methods then that is the diamond problem.

In java, in such situation code will not compile.

Solution to Diamond Problem:

Use interfacenName.super.methodName ()

**(Q) Why static methods are introduced in java8?**

(a)Only reason for introducing static methods in interface is that you can those methods with just interfacename.No need to create class and its object.

(b)Since interface can never contain:

. Constructors,

. Static blocks,

Nothing costly in terms of memory and performance.

(c)Hence we do not need to create object and hence if you have everting static, then for interface rather than class.

(d)You have this flexibility only after java8, before that you need to create class

**(Q)Are Static Methods available to implement classes by default?**

(a)Static Methods are not available to implementing classes.

(b)They are not default methods. They are static.

(c)Hence, you can call these methods using Interface Name explicitly from the implementing classes, as implementing classeswill not have access to these methods directly. (This is the disadvantage of static methods of interface that it is not available to implementing classes)

**(Q) What are predicates?**

(a) Predicate is a predefined functional interface (Having only one abstract method)

(b) The Only abstract method of predicate is test (T t)

Example: public boolean test (T t)

(c)Whenever we want to go for some Boolean condition then you can go for predicates.

**Type Parameter and return types of Predicates:**

(1)Input to predicate can be anything like

**Example:**

(a)Predicate<String> (b) Predicate<Integer> (c) Predicate<Employee>

(2) Hence only one type argument is required which is input type in predicate.

(3)Return Type is not required as it is always Boolean only.

**Advantages:**

(a)Code reusability.

(b)If you have same condition being used 100 times in a program then you can write once and just use 100 times with checkLength.test (different string to be tested)

(c)Conditional Checks are holded by functional interfaces.

**(Q) What is Predicate Joining?**

(a)You can combine predicates in serial predicate

(b) Three ways to join:

And

Or

Negate

Example: If you want to test two conditions:

1. To check length of string > 5
2. To check if length is even

**(Q) What are Functions?**

(a) Function is also a predefined Functional Interface (Having only one abstract method).

(b) The only abstract method of Function is apply (T t); R apply (T t);

(c) Given some input perform some operation on input and then produce / return result (not necessary a Boolean value).

This takes one input and returns one output. In predicate, we used to take one input and return type is always Boolean. In function return type is not fixed hence we declare both input type and return type

Both predicate and Function present in java.util.function package.

**(Q) What is the difference between Predicate and Function?**

**Predicate Function**

1. It has return type as Boolean, (a) It has return type as object. It is

It is used for conditional checks used to perform operations and return result

1. written: Predicate<T>,accepts single arg (b)written:Function<T,R> accepts single arg

However, return any type of obj denoted by R.

(c)Contains test () method (c) contains apply () method.

**(Q) What is Functional chaining?**

We can combine / chain multiple functions together with and Then.

(1)There are two ways to combine functions:

f1.andThen(f2).apply (Input); - first f1 then f2

f1.compose(f2).apply(Input) - first f2 then f1

Multiple functions can be chained together like:

f1.andThen (f2).andThen(f3).and Then(f4).apply(Inputs);

**(Q) What is Consumer Functional Interface?**

(a) Predicate<T> takes one input and return Boolean.

(b) Function <T, R> takes one input and one return type produced after performing some operation on that input.

(c) Consumer<T> -> it will consume Item. Consumers never return anything (never Supply), they just consume.

(d) Ex: take any object and save its details in database and do not return anything.

Interface Consumer<T> {

Public void accept (T t)

}

**(Q) What is consumer chaining?**

(a) We can combine / chain multiple consumers together with and Then.

(b) There is only one ways to combine consumers:

c1.andThen(c2).apply(Input); - first c1 and c2

1. No compose() in consumer.
2. c1.andThen(c2).andThen(c3).andThen(c4).apply(Inputs);

**(Q) What is Supplier Functional Interface?**

(a) Supplier<R> -> it will just supply required objects and will not take any input

(b) Its always going to supply never consume/ take any input.

(c) Ex: always supply me current date.

Interface Supplier<R> {

Public R get();

}

1. No chaining as no input is given to this. Only it gives you output.

**Advantages of Supplier**

1. Write once, use anywhere
2. Code reusability.

**(Q) Use of BiConsumer BiFunction, BiPredicate and Why no BiSupplier?**

Until now, we had:

Predicate<T> -> test () -> return Boolean

Function<T, R> -> apply () -> returns anything

Consumer<T> -> accept () -> returns nothing

Supplier<R> -> get () -> returns anything.

**(Q) What if we need two arguments for operation?**

(Ans)Then we needs Bi XYZ Functional Interfaces.

There is no input in supplier so no one 0r 2 input arguments needed. Hence no Bi Supplier.

**(Q) If we want to operate on three arguments then tripredicate?**

(a) There are no predicate or Trifunction etc.

(b) No Quad Predicate No Quad function.

(c) Java 8 has inbuilt functional interface can take only one or two arguments no more.

**(Q) What are streams?**

(a)If we want to process bulk objects of collection then go for streams concept.

(b)Way to operate on collection in java 8 is Stream.

(c)A special iterator class allows processing collections of objects in a functional manner.

Ex: fetch all objects from collection of list whose value is greater than 15

**(Q)** **Why streams were introduced in java 8 if we already had java.io.stream?**

(a) Java Io streams is a sequence of characters or binary data, which is used to be written to a file or to read data from a file.

(b) While streams java 1.8 is nowhere related to files, it is related to collection object.

(c) Java Io streams related to file whereas java 8 streams are related to collection object.

(d) Hence if we need to perform some operations on collection there we should go for streams.

**(Q) Difference between streams (1.8) and collection?**

(a) To represent group of collection as single entity then we should use collection concept.

(b) If we want to operation on bulk objects in collection then we should go for streams.

**(Q) Steps to Create and process stream?**

(a)We can get stream object by:

Stream s = collectionObject.stream();

(b)Once we get stream object we can process the object of collection.

(c) Processing of stream consists of two steps/stages

(i) Configuration of stream

(ii) Processing that configuration

Configuration can be done by Map and Filter.

**(Q) How to filter the stream objects?**

Stream s = collectionObject.stream ().filter (i -> i % 2 ==0)

**(Q) How to map Stream Object?**

(a) What if we do not want to filter out?

(b)We rather want to create a new object against each existing stream object based on some function

Ex: In given stream create new object by squaring its value?

**(Q) Difference between Filter and map?**

(a) If we want to fetch/filter objects from collection like

Ex: filter only even numbers from array list.

(b)If we want to perform, some operations on each objects of the collection then create another mapped object with different value (after operation is performed) for each object of that collection, then use map.

(c)In filter, because of filtering, number of objects in filtered list is less than the original list.

While in map, same number of objects are created in both new and original list created.

**(Q) Steps to create and process stream?**

**Streams:**

**Configuration of Stream:** Map and filter

**Processing that configuration:**

(a)Collect (b) Count (c) Sorted (d) Min Max (e) For Each (f) to Array (g) of

**Stream Processing**

**(Q)How To Process elements using Collect Method?**

**Collect**: If we want to collect elements of stream after filtering or mapping and add them to the required collection then use collect method.

**(Q)How To Process elements using count Method?**

**Count:** If we want to count how many elements are there in collection that satisfy given condition then use collect method.

**(Q) How to process elements using sorted method?**

**Sorted:**

1. If we want to sort elements inside a stream use this sorted () method.
2. We can sort based on default natural sorting order.
3. If we want to sort using customized sorting order then use comparator.

**(Q) How to process elements using sorted in descending order?**

**Comparator:**

1. If we want to sort using customized sorting order then use comparator.

**(Q)How to process elements using Min, Max?**

**Min (Comparator):** will return the minimum value based on the defined comparator.

**Max (Comparator):** will return the maximum value based on the defined comparator.

**(Q) How to process elements using Foreach?**

ForEach () is a method

1. All methods that we saw until now returned something, like min max value, sorted collection, etc.

(b)This method does not return anything. Rather this method takes lambda expression as argument and apply that lambda expression to each element present in that stream

**(Q) How to process elements using toArray ()?**

We can use this method to copy elements present in the stream to specified array.

**(Q) How to process elements using of ()?**

(a) Stream concept is not applicable just for the collections it is also applicable for any group of value.

(b) Even for arrays you can use stream.

(c) Stream. Of () this method can take any group of values and convert them to stream so that multiple stream operations can be applied to it.

**(Q) What is a parallel stream?**

(a) Java Parallel Streams came into picture after java 1.8.

(b) Its means to utilize multiple cores of processor.

(c) Till now our java code has one stream of processing where it executes sequentially.

(d)But when you use parallel streams, we divide code into multiple streams that executes parallel, On Separator cores and the final result is the outcome of individual cores outcomes combined

Sequential Stream?

Tasks Core Task 1 Task 2 Task3 Task4

T1 CORE1 T1 T2 T3 T4

T2 CORE2

T3 CORE3

T4 CORE4

The output of the this sequential stream is T1,T2,T3,T4 -> In sequential order tasks are executed and output of 1 can be input to another.

Parallel Stream?

Tasks Core Task 1 Task 2 Task3 Task4

T1 CORE1 T2

T2 CORE2 T4

T3 CORE3 T1

T4 CORE4 T3

1. The output of this parallel stream is T2,T4,T1,T3
2. Order of execution is not under control.
3. Hence, it is advisable to use parallel stream only when order of execution of threads does not matter and state of one element does not affect another.

Java Brains:

**Code in oop**

(1) Everything is an object

(2) All Code blocks are associated with classes and objects.

Passing lambda expressions:

Function as value:

Difference between anonymous inner class and interface in java?

No need to write { }, Incase if your lambda expression is one line.

**Example:**

**Java7 writing:**

**Public** **void** greet () {

System.***out***.println("Hello world");

}

Needs to assign it to a variable

greetingFunction = () -> { System.***out***.println("Hello world");}

**Java8 Writing:**

greetingFunction = () -> System.***out***.println("Hello world");

Method Having argument:

**public int** doubleNumber (int a) {

return a\*2;

}

It is actually invalid to specify the return keyword when you have a one – liner lambda expression without {}

doubleNumberFunction = (int a) -> a\*2;

addFunction = (int a, int b) -> a + b;

safeDivideFunction = (int a, int b) -> {

if(b == 0)

return 0;

return a/b;

};

stringLengthCountFunction = (String s) -> s.length();

**Type Inference:**

AnnonymousInnerclass in java