**Java8\_Feature\_Interview\_Questions**

**Q1) How do we avoid null pointer exception**

**A)** By using optional class we can avoid the null pointer exception. By using optional class we can specify alternative values to return or alternative code to run.

**Q2) What are the different keywords in optional class**

**A)** There are many methods in optional class, please find few of below

i.equals() ii.get() iii.empty() iv.hascode() v.filter() vi.ifPresent() vii.isPresent()

viii.map() ix.flatmap() x.tostring()

**Q3) What is the functional interface**

**A)** An interface that contains exactly one abstract method is known as functional interface.

It can have any number of default and static methods but can contain only one abstract method.

Java provides predefined functional interface to deal with functional programming by using lambda expression and method reference.

Java provide an annotation @FunctionalInterface which is used to declare and interface as functional interface

|  |  |
| --- | --- |
| **Interface** | **Description** |
| Consumer<T> | It represents an operator that accepts a single argument and no result  Ex: Consumer<String> consumer1 = ConsumerInterfaceExample::printMessage; |
| BiConsumer<T,U> | It represents an operator that accepts a two argument and no result  Ex: BiConsumer<Map<Integer, String>, String> biCon = BiConsumerInterfaceExample::ShowDetails; |
| Funtion<T,R> | It represent a function that accepts one argument and return a result  Ex:  Function<String, String> fun = FunctionInterfaceExample::show; |
| BiFuntion<T,U,R> | It represent a function that accepts two argument and return a result |
| Predicate<T> | It Represents a predicate <Boolean-value function> of one argument  Ex: Predicate<Integer> predicate =  PredicateInterfaceExample::checkAge; |
| BiPredicate<T,U> | It Represents a predicate <Boolean-value function> of two argument |

**Q4) What is lambda expression**

**A)** Lambda expression is a new feature in java-8. It is an anonymous function, it is used to implement the functional programming in java.

The lambda expression is used to provide the implementation of an interface which has the functional interface. It saves the lot of codes.

Whereas forms of writing Lambda expressions.

()-> expression

Ex: () -> {

//Body of no parameter lambda

}

(parameters) -> expression

(parameters) -> {multiple statements}

**Q5) What is stream API**

**A)** A stream API is used to process the collection objects. Streams don’t change the original data; they only provide the results as per the pipelined methods.

Stream does not store the elements; it simply conveys the elements from the source.

There are two operations in streams

1.Intermediate operations

* Map
* Filter
* Sort

2.Terminal operators

* Collect
* forEach
* reduce

**Q6) What are the Java-8 features**

**A)** There are many new features are introduced in java-8

i. Lambda Expressions ii. Method references

iii. Functional Interfaces iv. Stream API

v. Default Methods vi. Static Methods in Interfaces

vii. optional classes viii. Collector class

ix. Time API x. ForEach

**Q7) What is an optional class and how do we use it.**

**A)** Every java programmer is familiar with null pointer exception. It can crash our code and it is very hard to avoid it without using many null checks. So, to overcome these java-8 has introduced a new class called optional class in java.util package. By using optional class, we can specify alternative values to return or alternative code to run.

There are many methods in optional class to overcome null pointer exception

1. Empty() ii. Of()

iii. ofNullable() iv. Get()

v. isPresent() vi. ifPresent()

vii. filter() viii. Map()

ix. flatMap() x. sort()

**Q8) What are the thread safe collections**

**A)** Concurrent hashmap and copyonrightarray

**Q9) What is Method reference**

**A)** Method reference is used to refer the method of functional interfaces. It is compact and easy form of lambda expressions. Each time when we are using lambda expression to just referring a method, we can replace lambda expression with method reference.

There are three ways we can do method reference

1. Reference to a static method

EX: ContainingClass::staticMethodName

1. Reference to a instance method

Ex: containingObject::instanceMethodName

1. Reference to a constructor

Ex: ClassName::new

**Q10) What is difference between array and arrayList**

**A)** The major different between array and arrayList is the static nature of array and dynamic nature of arrayList.

Once created you cannot alter the size of an array whereas an arrayList can re-size itself as and when required.

Another difference is that and array is basic functionality provided by java on the other side arrayList is a part of collection framework.

**Q11) What is default methods**

**A)** Interface could have only abstract methods the implementation of these methods has to be provided in a separate class So, if a new method is to be added in an interface then it’s implementation code has to be provided in the class implementing the same interface.

To overcome this issue, java-8 has introduced the concept of default methods which allow the interface to have method with implementation without affecting the classes that the interface.

**Q12) How to find second largest number in array by using stream API.**

**A)** int[] numbers={5,9,11,2,8,21,1}

System.out.println("numbers in array:"+arrays.toString(numbers));

Ex-1:

Int secondLargestnumber=Arrays.stream(number).boxed().sorted(comparator.reverseorder()).limit(2).skip(1).findfirst().get();

Ex-2:

Int secondLargestnumber=numbers.stream().sorted(comparator.revereseorder()).skirp(1).findfirst().get();

**Q13)How to find out employee’s avg salary by using stream API.**

**A)** List<Employee> list=Arrays.asList(employees);

Double avg=list.stream().mapToDouble(Employee::getsalary()).average().getAsDouble();

**Q14) How to find even numbers employees by using streams**

**A)** Even numbers:

List<integer> evenemployees=list.stream. filter(e->e%2==0).collect(Collectors.toList());

Odd numbers:

List<Integers> oddemployees=list.stream. filter(e->e%2!=0).collect(Collectors.toList())

**Q15) Why static method introduced in interface**

**A)** If you want provide default implementation which cannot be changes, we can use static methods in interface. Static method cannot be overridden.

**Q16) What is the advantage of using new Date API.**

**A)** New date-time API is introduced in Java 8 to overcome the following drawbacks of old date-time API :

1. **Not thread safe :** Unlike old java.util.Date which is not thread safe the new date-time API is immutable and doesn’t have setter methods.
2. **Less operations :** In old API there are only few date operations but the new API provides us with many date operations.

Java 8 under the package java.time introduced a new date-time API, most important classes among them are :

1. **Local :** Simplified date-time API with no complexity of timezone handling.
2. **Zoned :** Specialized date-time API to deal with various timezones.

**Q17) What is difference between collection API and stream API**

**A)** Collection API:

* Elements are not computed on demand basis
* Index base access possible
* Iteration was external
* Access of elements was sequential

Stream API:

* Elements are computed on demand basis
* Index base access are not possible
* Iteration is internal
* Access of elements could be sequential as well as parallel
* Stream doesn’t store data
* An operation on stream does not change its original data
* Streams doesn’t have a fixed size

**Q18) In java-8 how meta space difference from the permgen**

**A)** PermGen is the memory area for storing class data like static variable,byte code and etc. By default 64 Mb is allocated for PermGen. It can be tuned by using -XXMaxPermSize.

In Java 8, PermGen method area replaced with MetaSpace. They have moved permGem to the separate memory in the native OS and that is called MetaSpace. It can by default auto increases its size. In MetaSpace, classes can load and unload during the lifespan of the JVM.

| **Sr. No.** | **Key** | **PermGen** | **MetaSpace** |
| --- | --- | --- | --- |
| 1 | Basic | PermGen is the memory area for storing class data like static variable,byte code and etc | In Java 8, PermGen method area replaced with MetaSpace |
| 2 | Default Memory Allocation | By default 64 Mb is allocated for PermGen | It can by default auto increases its size |
| 3 | Tuned-up Memory Flag | It can be tuned by using -XXMaxPermSize. | We can restrict upper bound of the memory by -XX:MaxMetaspaceSize |
| 4 | Memory Area | It is a special Heap space. | Since Java 8, It is now separate memory area  in the native OS |