Java 8

350. What are the new features released in Java 8?

The new features released in Java 8 are:

1. Lambda Expression
2. Stream API
3. Date and Time API
4. Functional Interface
5. Interface Default and Static Methods
6. Optional
7. Base64 Encoding and Decoding
8. Nashorn JavaScript Engine
9. Collections API Enhancements
10. Concurrency Enhancements
11. Fork/Join Framework Enhancements
12. Spliterator
13. Internal Iteration
14. Type Annotations and Repeatable Annotations
15. Method Parameter Reflection
16. JVM Parameter Changes

351. What are the main benefits of new features introduced in Java 8?

The main benefits of Java 8 features are:

1. Support for functional programming by Lambda and Streams
2. Ease of high volume data processing by Streams
3. Ease of use by getting Parameter names through Reflection
4. Reusable code with enhanced Collection APIs
5. Smart exception handling with Optional
6. Control on JVM with new Parameters
7. Enhanced encryption support with Base 64
8. Faster execution with Nashorn JavaScript engine support

352. What is a Lambda expression in Java 8?

Lambda expression is an anonymous function. It is like a method that does not need any access modifiers, name or return value declaration. It accepts a set of input parameters and returns result.

Lambda expression can be passed as a parameter in a method. So we can treat code in Lambda expression as data. This piece of code can be passed to other objects and methods.

353. What are the three main parts of a Lambda expression in Java?

Three main parts of a Lambda expression are:

1. Parameter list: A Lambda expression can have zero or more parameters. Parameter list is optional to Lambda.

2. Lambda arrow operator: "->" is known as Lambda arrow operator. It separates the list of parameters and the body of Lambda.

3. Lambda expression body: The piece of code that we want to execute is written in Lambda expression body.

E.g. In following example:
Arrays.asList("a", "b", "d").forEach(e -> System.out.println(e));
Parameter list = e
Arrow = ->
Body = System.out.println(e)

354. What is the data type of a Lambda expression?

A Lambda expression fulfills the purpose of passing code as data.

The data type of a Lambda expression is a Functional interface.

In most of the cases this is java.lang.Runnable interface.

355. What is the meaning of following lambda expression?

(e -> System.out.println(e));

This Lambda expression takes a parameter e and prints it via System.out.

356. Why did Oracle release a new version of Java like Java 8?

The main theme of Java 8 is support for functional programming. With increase in Database size and growth of multi-code CPU servers, there is need for Java to support such large-scale systems.

With new features of Java 8, it is possible to create functional programs to interact efficiently with Big Data systems. Support for Streams is very helpful in this regard.

Lambda expressions are very useful for cloud computing where we can pass code as data and run the same code on multiple servers.

Optional is a best practice that is borrowed from Google Guava library for handling the exceptional cases. This has made programs more robust with support for edge cases.

357. What are the advantages of a lambda expression?

We can pass a lambda expression as an object to a method. This reduces the overhead involved in passing an anonymous class.

We can also pass a method as a parameter to another method using lambda expressions.

358. What is a Functional interface in Java 8?

A Functional interface in Java is an interface that has exactly one abstract method.

It can have default methods with implementation. A default method is not abstract.

In Java 8, java.lang.Runnable and java.util.concurrent.Callable are two very popular Functional interfaces.

359. What is a Single Abstract Method (SAM) interface in Java 8?

A Functional interface is also known as Single Abstract Method Interface, since it has exactly one abstract method.

360. How can we define a Functional interface in Java 8?

To define a Functional interface in Java 8, we can create an Interface with exactly one abstract method.

Another way is to mark an Interface with annotation @FunctionalInterface. Even with the annotation we have to follow the rule of exactly one abstract method.

The only exception to this rule is that if we override java.lang.Object class's method as an abstract method, then it does not count as an abstract method.

361. Why do we need Functional interface in Java?

Functional Interfaces are mainly used in Lambda expressions, Method reference and constructor references.

In functional programming, code can be treated as data. For this purpose Lambda expressions are introduced. They can be used to pass a block of code to another method or object.

Functional Interface serves as a data type for Lambda expressions. Since a Functional interface contains only one abstract method, the implementation of that method becomes the code that gets passed as an argument to another method.

362. Is it mandatory to use @FunctionalInterface annotation to define a Functional interface in Java 8?

No, it is not mandatory to mark a Functional interface with @FunctionalInterface annotation.

Java does not impose this rule.

But, if we mark an interface with @FunctionalInterface annotation then Java Compiler will give us error in case we define more than one abstract method inside that interface.

363. What are the differences between Collection and Stream API in Java 8?

Main differences between Collection and Stream API in Java 8 are:

1. Version: Collection API is in use since Java 1.2. Stream API is recent addition to Java in version 8.
2. Usage: Collection API is used for storing data in different kinds of data structures. Stream API is used for computation of data on a large set of Objects.
3. Finite: With Collection API we can store a finite number of elements in a data structure. With Stream API, we can handle streams of data that can contain infinite number of elements.
4. Eager vs. Lazy: Collection API constructs objects in an eager manner. Stream API creates objects in a lazy manner.
5. Multiple consumption: Most of the Collection APIs support iteration and consumption of elements multiple times. With Stream API we can consume or iterate elements only once.

364. What are the main uses of Stream API in Java 8?

Main uses of Stream API in Java 8 are:

1. It helps in using data in a declarative way. We can make use of Database functions like Max, Min etc., without running a full iteration.
2. It makes good use of multi-core architectures without worrying about multi-threading code.
3. We can create a pipeline of data operations with Java Stream that can run in a sequence or in parallel.
4. It provides support for group by, order by etc. operations.
5. It supports writing for code in Functional programming style.
6. It provides parallel processing of data.

365. What are the differences between Intermediate and Terminal Operations in Java 8 Streams?

Main differences between Intermediate and Terminal Stream operations are as follows:

1. Evaluation: Intermediate operations are not evaluated until we chain it with a Terminal Operation of Stream. Terminal Operations can be independently evaluated.
2. Output: The output of Intermediate Operations is another Stream. The output of Terminal Operations is not a Stream.
3. Lazy: Intermediate Operations are evaluated in lazy manner. Terminal Operations are evaluated in eager manner.
4. Chaining: We can chain multiple Intermediate Operations in a Stream. Terminal Operations cannot be chained multiple times.
5. Multiple: There can be multiple Intermediate operations in a Stream operation. There can be only one Terminal operation in Stream processing statement.

366. What is a Spliterator in Java 8?

A Spliterator is a special type of Iterator to traverse and partition the elements of a source in Java. A source can be a collection, an IO channel or a generator function.

A Spliterator may traverse elements individually or sequentially in bulk.

367. What are the differences between Iterator and Spliterator in Java 8?

Main differences between Iterator and Spliterator are as follows:

1. Spliterator can be used with Streams in Java 8. Where as, Iterator is just used with Collection.
2. Spliterator uses Internal Iteration to iterate Streams. Iterator uses External Iteration to iterate Collections.
3. Spliterator can iterate Streams in Parallel as well as Sequential manner. Iterator only iterates in Sequential manner.
4. Spliterator can traverse elements individually as well as in bulk. Iterator only iterates elements individually.

368. What is Type Inference in Java 8?

A Java compiler can see each method's invocation and it declaration to determine what are type arguments required for invocation.

By Type Inference, Java can determine the types of the arguments as well as the type of the result being returned.

Type inference algorithm also tries to find the most specific type that can work with all types of arguments.

369. Does Java 7 support Type Inference?

Yes, Java 7 supports Type Inference. In Java 8, Oracle has enhanced the Type Inference concept. Now it can be used to define Lambda expressions, functions and Method references.

370. How does Internal Iteration work in Java 8?

In an Iterator, the fundamental question is that which party controls the iteration. Is it Iterator or the Collection on which iterator runs.

When a Collection controls the iterator, then it is called External Iteration. When the Iterator controls the iteration then it is called Internal Iteration.

In case of Internal Iteration, the client hands over an operation to Iterator and the Iterator applies the operation to all the elements in aggregate.

Internal Iteration is easier to implement, since the Iterator does not have to store the state of the collection.

371. What are the main differences between Internal and External Iterator?

Main differences between Internal and External Iterator are as follows:

1. An Internal Iterator controls the iteration itself. In an External Iterator collection controls the iteration.
2. Internal Iterator can iterate elements in individually as well as in
3. Bulk (like forEach). External iterator iterates element one by one.
4. Internal Iterator does not have to iterate elements only sequentially. External Iterator always iterates sequentially.
5. Internal Iterator supports declarative programming style that goes well with functional programming. External Iterator follows imperative style OOPS programming.
6. Some people consider Internal Iterator code more readable than that of External Iterator.

372. What are the main advantages of Internal Iterator over External Iterator in Java 8?

Some of the main advantages of Internal Iterator are:

1. Internal Iterator is based on Functional programming, therefore it can work on declarative style code.
2. There is no need to sequentially iterate elements in Internal Iterator.
3. Code is more readable and concise in Internal Iterator.
4. Internal Iterator supports concurrency and parallel processing.

373. What are the applications in which we should use Internal Iteration?

We need Internal Iterator in applications that require high performance, parallel processing, fast iteration and bulk operations support.
Also in Internal Iteration applications, we do not have much control over iteration. The other features like parallel processing etc. become more important.

374. What is the main disadvantage of Internal Iteration over External Iteration?

Internal Iteration has many advantages over External Iteration. But it has one big disadvantage. Since Java API is responsible for iterating in Internal iterator, developer does not get any control over iteration.

375. Can we provide implementation of a method in a Java Interface?

Before Java 8, it was not allowed to provide implementation of a method in an Interface.
Java 8 has introduced the flexibility of providing implementation of a method in an interface. There are two options for that:

1. Default Method: We can give default implementation of a method.
2. Static Method: We can create a static method in an interface and provide implementation.

376. What is a Default Method in an Interface?

In Java 8, we can provide implementation of a method in an Interface and mark this method with Default keyword.

In this way, this implementation of the method becomes default behavior for any class implementing the interface.

377. Why do we need Default method in a Java 8 Interface?

Default methods in an Interface provide backward compatibility feature in Java 8.

Let say there is an interface Car that is implemented by BMW, Chevrolet and Toyota classes. Now a Car needs to add capability for flying. It will require change in Car interface. Some of the car classes that do not have flying capability may fail. Therefore a Default Implementation of flying methods is added in Car interface so that cars with no flying capability can continue to implement the original Car interface.

378. What is the purpose of a Static method in an Interface in Java 8?

A Static method in an Interface is utility or helper method. This is not an object level instance method. Some of the uses of Static method in an Interface are:

1. Single Class: There is no need to create a separate Utils class for storing utility or helper methods. We can keep these methods in same interface.

2. Encapsulation: With Static methods, complete behavior of a Class is encapsulated in same class. There is no need to maintain multiple classes.
3. Extension: It is easier to extend a Class/API. If we extend a collection ArrayList, we get all the methods. We need not extend Collections class also.

379. What are the core ideas behind the Date/Time API of Java 8?

There are three core ideas behind the Date/Time API of Java 8:

1. Immutable-value classes: The new API avoids thread-safety and concurrency issues by ensuring that all the core classes are immutable and represent well-defined values.

2. Domain-driven design: The new API is modeled on precise domain with classes that represent different use cases for Date and Time.

3. The emphasis on domain-driven design offers benefits like clarity and understandability.

4. Separation of chronologies: The new API allows people to work with different calendar systems. It supports the needs of users in different areas of the world likes Japan or Thailand that don't follow ISO-8601.

380. What are the advantages of new Date and Time API in Java 8 over old Date API?

Some of the advantages of Java 8 Date Time API over existing Date API are:

Concurrency: Existing Date Time classes (such as java.util.Date and SimpleDateFormatter) are not thread-safe. This does not work well in concurrent applications. In new Date Time API, developer does not have to deal with concurrency issues while writing date-handling code.

Better Design: Date/Time classes prior to Java 8 have poor API design. For example, years in java.util.Date start at 1900, months start at 1, and days start at 0. It is not very intuitive. Java 8 Date Time API handles it very well.

No need for 3rd Party Libraries: With the popularity of third-party Date/Time libraries like Joda Time, Java has to make its native Date/Time API comparable. Now we can use the Java API instead of using 3rd party libraries.

381. What are the main differences between legacy Date/Time API in Java and Date/Time API of Java 8?

Main difference between legacy Date/Time API and Java 8 Date/Time API are:

1. Old API is not Thread safe. Java 8 API is Thread safe.
2. Old API has many mutable objects. New Java 8 API is based on Immutable objects.
3. Performance of old API is not good. New Java 8 Date/Time API gives better performance.
4. Old API is less readable and maintainable. New Java 8 API is very well designed and is more readable.
5. Old API has month values from 0 to 11. New API has months from 1 to 12.

382. How can we get duration between two dates or time in Java 8?

In Java8, we have a new class Duration that provides the utility of computing duration between two dates.

We can call the static method Duration.between(date1, date2) to get the time period in hours, mins, days etc. between date1 and date2.

383. What is the new method family introduced in Java 8 for processing of Arrays on multi core machines?

Java 8 has enhanced the Arrays class with methods that can run efficiently on multi core machines.

These methods start with keyword parallel.
Egg. Arrays.parallelSetAll(), Arrays.parallelSort() etc.

This parallel set of methods provides parallel processing of Arrays that can run Java code very fast on a multi core machine.

384. How does Java 8 solve Diamond problem of Multiple Inheritance?

In Multiple Inheritance if a class extends more than one classes with two different implementation of same method then it causes Diamond problem.
Consider following example to see problem and solution for Diamond problem in Java 8:

```
public interface BaseInterface{
  default void display() { //code goes here }
}
public interface BaseOne extends BaseInterface { }
public interface BaseTwo extends BaseInterface { }
public class ChildClass implements BaseOne, BaseTwo { }
```

In the above code, class ChildClass gives compile time error. Java Compiler cannot decide which display method should it invoke in ChildClass.
To solve this problem, Java SE 8 has given the following remedy:

```
public interface A{
  default void display() { //code goes here }
}
public interface B extends A{ }
public interface C extends A{ }
public class D implements B,C{
  default void display() {
    B.super.display();
  }
}

public interface BaseInterface{
  default void display() { //code goes here }
}
public interface BaseOne extends BaseInterface { } public interface
BaseTwo extends BaseInterface { } public class ChildClass
implements BaseOne, BaseTwo {
  default void display(){
  BaseOne.super.display();
}
}
```

The method invocation at BaseOne.super.display(); solves the Diamond problem as it resolves the confusion for compiler.

385. What are the differences between Predicate, Supplier and Consumer in Java 8?

The subtle difference between Predicate, Supplier and Consumer in Java 8 is as follows:

Predicate is an anonymous function that accepts one argument and returns a result.

Supplier is an anonymous function that accepts no argument and returns a result.

Consumer is an anonymous function that accepts one argument and returns no result.

386. Is it possible to have default method definition in an interface without marking it with default keyword?

No, we have to always mark a default method in interface with default keyword.

If we create a method with implementation in an interface, but do not mark it as default, then we will get compile time error.

387. Can we create a class that implements two Interfaces with default methods of same name and signature?

No, it is not allowed to create a class that implements interfaces with same name default methods.

It will give us compile time error for duplicate default methods.

388. How Java 8 supports Multiple Inheritance?

In Multiple Inheritance a class can inherit behavior from more than one parent classes.
Prior to Java 8, a class can implement multiple interfaces but extend only one class.

In Java 8, we can have method implementation within an interface.
So an interface behaves like an Abstract class.

Now if we implement more than one interface with method implementation in a class, it means we are inheriting behavior from multiple abstract classes. That is how we get Multiple Inheritance in Java 8.

389. In case we create a class that extends a base class and implements an interface. If both base class and interface have a default method with same name and arguments, then which definition will be picked by JVM?

In such a scenario, JVM will pick the definition in base class.

390. If we create same method and define it in a class , in its parent class and in an interface implemented by the class, then definition will be invoked if we access it using the reference of Interface and the object of class?

In all the cases, method defined in the class will be invoked.

391. Can we access a static method of an interface by using reference of the interface?

No, a static method of interface has to be invoked by using the name of the interface.

392. How can you get the name of Parameter in Java by using reflection?

Java 8 has introduced a method Parameter.getName() to get the name of a parameter by using reflection.

Before using this feature, we need to turn on this feature in Java compiler.

To turn on this feature, just run javac with –parameters argument.

To verify the availability of this feature, we can use Parameter. isNamePresent() method.

393. What is Optional in Java 8?

Optional is a container object that may have a null or non-null value. If it has a value then isPresent() method returns true.

It a value is present, we can call get() method to get the value. Else we will get nothing.

It is very useful in handling data that has null values.

394. What are the uses of Optional?

Some of the uses of Optional in Java are:

We can use Optional to avoid NullPointerException in an application.
Optional performs Null check at compile time, so we do not get run time exception for a null value.
Optional reduces the codebase pollution by removing unnecessary null checks.
Optional can also be used to handle default case for data when a value is null.

395. Which method in Optional provides the fallback mechanism in case of null value?

In case, an Optional has null value, we can use orElseGet() method as fallback mechanism. If we implement orElseGet() method, it will be invoked when the value of Optional is null.

396. How can we get current time by using Date/Time API of Java 8?

In Java 8 we can use Clock class to get the current time. Instead of using old method System.currentTimeMillis(), we can create a Clock object and call millis() method to get the current time in milliseconds.

We can also call instant() method on Clock object to get the current time in a readable format.

397. Is it possible to define a static method in an Interface?

Yes, from Java 8, an Interface can also have a static method.

398. How can we analyze the dependencies in Java classes and packages?

Java 8 comes with a new command line tool jdeps that can help in analyzing the package-level and class-level dependencies.

We can pass a jar file name or a class name as an argument to this tool. It will list all the dependencies of that jar or class.

399. What are the new JVM arguments introduced by Java 8?

In Java 8, PermGen space of ClassLoader is removed. It has been replaced with MetaSpace.

Now we can set the initial and maximum size of MetaSpace.

The JVM options -XX:PermSize and –XX:MaxPermSize are replaced by -XX:MetaSpaceSize and -XX:MaxMetaspaceSize respectively in Java 8.

400. What are the popular annotations introduced in Java 8?

Some of the popular annotations introduced in Java 8 are:

@FunctionalInterface: This annotation is used to mark an interface as Functional Interface. As mentioned earlier, A FunctionalInterface can be used for lambda expressions.

@Repeatable: This annotation is used for marking another annotation. It indicates that the marked annotation can be applied multiple times on a type.

401. What is a StringJoiner in Java 8?

StringJoiner is a new class in Java 8 that can be used to create a String. It can construct a sequence of characters separated by a delimiter. It can also optionally add a prefix and suffix to this sequence. We can use this sequence to get a String.

E.g.
The String "[One:Two:Three]" may be constructed as follows:

```
StringJoiner sj = new StringJoiner(":", "[", "]");
sj.add("One").add("Two").add("Three"); String
desiredString = sj.toString();
```

402. What is the type of a Lambda expression in Java 8?

The type of a lambda expression depends on the context it is being used.

A lambda is like a method reference. It does not have a type of its own.

Generally, a Lambda is an instance of a Functional Interface.

403. What is the target type of a lambda expression ?

The target type of a lambda expression represents a type to which the expression can be converted.

The target type for a lambda expression is a functional interface.

The lambda expression must have same parameter type as the parameter in the function of the interface. It must also return a type compatible with the return type of function.

404. What are the main differences between an interface with default method and an abstract class in Java 8?

An interface with a default method appears same as an Abstract class in Java. But there are subtle differences between two.

1. Instance variable: An interface cannot have instance variables. An abstract class can have instance variables.
2. Constructor: An interface cannot have a constructor. An abstract class can have constructor.
3. Concrete Method: An interface cannot have concrete methods other than default method. An abstract class is allowed to define concrete methods with implementation.
4. Lambda: An interface with exactly one default method can be used for lambda expression. An abstract class cannot be used for lambda expression.