**Why do we need change to Java again?**

Oracle Corporation has introduced a lot of new concepts in Java SE 8 to introduce the following benefits:

* **To Utilize Current Multi-Core CPUs Efficiently**
* Recently, we can observe drastic changes in Hardware. Now-a-days, all systems are using Multi-Core CPUs (2,4,8,16-Core etc.) to deploy and run their Applications. We need new Programming Constructs in Java to utilize these Multi-Core Processors efficiently to develop Highly Concurrently and Highly Scalable applications.
* **To Utilize FP Features**
* Oracle Corporation has introduced a lot of FP(Functional Programming) concepts as part of Java SE 8 to utilize the advantages of FP.

**Java SE 8 New Features?**

* Lambda Expressions
* Functional Interfaces
* [Stream API](https://www.journaldev.com/2774/java-8-stream)
* Date and Time API
* [Interface Default Methods and Static Methods](https://www.journaldev.com/2752/java-8-interface-changes-static-method-default-method)
* Spliterator
* Method and Constructor References
* Collections API Enhancements
* Concurrency Utils Enhancements
* Fork/Join Framework Enhancements
* Internal Iteration
* Parallel Array and Parallel Collection Operations
* Optional
* Type Annotations and Repeatable Annotations
* Method Parameter Reflection
* Base64 Encoding and Decoding
* IO and NIO2 Enhancements
* Nashorn JavaScript Engine
* javac Enhancements
* JVM Changes
* Java 8 Compact Profiles: compact1,compact2,compact3
* JDBC 4.2
* JAXP 1.6
* Java DB 10.10
* Networking
* Security Changes

**Advantages of Java SE 8 New Features?**

We can get the following benefits from Java SE 8 New Features:

* More Concise and Readable code
* More Reusable code
* More Testable and Maintainable Code
* Highly Concurrent and Highly Scalable Code
* Write Parallel Code
* Write Database Like Operations
* Better Performance Applications
* More Productive code

**What is Lambda Expression?**

Lambda Expression is an anonymous function, which accepts a set of input parameters and returns results.

Lambda Expression is a block of code without any name, with or without parameters and with or without results. This block of code is executed on demand.

**What are the three parts of a Lambda Expression? What is the type of Lambda Expression?**

A Lambda Expression contains 3 parts:

* + - * Parameter List

A Lambda Expression can contain zero or one or more parameters. It is optional.

* Lambda Arrow Operator
  + - * 1. “->” is known as Lambda Arrow operator. It separates parameters list and body.
* Lambda Expression Body

The type of “Journal Dev” is java.lang.String. The type of “true” is Boolean. In the same way, what is the type of a Lambda Expression?

The Type of a Lambda Expression is a [Functional Interface](https://www.journaldev.com/2763/java-8-functional-interfaces).

Example:- What is the type of the following Lambda Expression?

()->System.out.println("Hello World");

This Lambda Expression does not have parameters and does return any results. So it’s type is “java.lang.Runnable” Functional Interface.

**What is a Functional Interface? What is SAM Interface?**

A Functional Interface is an interface, which contains one and only one abstract method. Functional Interface is also know as SAM Interface because it contains only one abstract method.

SAM Interface stands for Single Abstract Method Interface. Java SE 8 API has defined many Functional Interfaces.

**Is is possible to define our own Functional Interface? What is @FunctionalInterface? What are the rules to define a Functional Interface?**

Yes, it is possible to define our own Functional Interfaces. We use Java SE 8’s @FunctionalInterface annotation to mark an interface as Functional Interface.

We need to follow these rules to define a Functional Interface:

* Define an interface with one and only one abstract method.
* We cannot define more than one abstract method.
* Use @FunctionalInterface annotation in interface definition.
* We can define any number of other methods like Default methods, Static methods.
* If we override java.lang.Object class’s method as an abstract method, which does not count as an abstract method.

**Is @FunctionalInterface annotation mandatory to define a Functional Interface? What is the use of @FunctionalInterface annotation? Why do we need Functional Interfaces in Java?**

It is not mandatory to define a Functional Interface with @FunctionalInterface annotation. If we don’t want, We can omit this annotation. However, if we use it in Functional Interface definition, Java Compiler forces to use one and only one abstract method inside that interface.

Why do we need Functional Interfaces? The type of a Java SE 8’s Lambda Expression is a Functional Interface. Whereever we use Lambda Expressions that means we are using Functional Interfaces.

**When do we go for Java 8 Stream API? Why do we need to use Java 8 Stream API in our projects?**

When our Java project wants to perform the following operations, it’s better to use [Java 8 Stream](https://www.journaldev.com/2774/java-8-stream) API to get lot of benefits:

* When we want perform Database like Operations. For instance, we want perform groupby operation, orderby operation etc.
* When want to Perform operations Lazily.
* When we want to write Functional Style programming.
* When we want to perform Parallel Operations.
* When want to use Internal Iteration
* When we want to perform Pipelining operations.
* When we want to achieve better performance.

**Explain Differences between Collection API and Stream API?**

|  |  |  |
| --- | --- | --- |
| S.NO. | COLLECTION API | STREAM API |
| 1. | It’s available since Java 1.2 | It is introduced in Java SE8 |
| 2. | It is used to store Data(A set of Objects). | It is used to compute data(Computation on a set of Objects). |
| 3. | We can use both Spliterator and Iterator to iterate elements. We can use [forEach](https://www.journaldev.com/13941/java-foreach-java-8-foreach) to performs an action for each element of this stream. | We can’t use Spliterator or Iterator to iterate elements. |
| 4. | It is used to store limited number of Elements. | It is used to store either Limited or Infinite Number of Elements. |
| 5. | Typically, it uses Internal Iteration concept to iterate Elements. | It uses External Iteration to iterate Elements. |
| 6. | Collection Object is constructed Eagerly. | Stream Object is constructed Lazily. |
| 7. | We add elements to Collection object only after it is computed completely. | We can add elements to Stream Object without any prior computation. That means Stream objects are computed on-demand. |
| 8. | We can iterate and consume elements from a Collection Object at any number of times. | We can iterate and consume elements from a Stream Object only once. |

**What is Spliterator in Java SE 8?Differences between Iterator and Spliterator in Java SE 8?**

Spliterator stands for Splitable Iterator. It is newly introduced by Oracle Corporation as part Java SE 8.

Like Iterator and ListIterator, It is also one of the Iterator interface.

* **1.**    [**Create a Website**](http://625827376.keywordblocks.com/Create_a_Website.cfm?&vi=1508243580556658689&dytm=1508243581691&rtbsd=10&npgv=1&kbbq=&sde=1&adepth=1&ddepth=1&tdAdd%5B%5D=%7C@%7Csde=1%7C@%7Cadepth=1%7C@%7Cddepth=1&verid=121199&hvsid=00001508243580966002962362144158&upk=1508243581.2664&sttm=1508243580966&=&vgd_bdata=url_l=4%7C%7Cdevice_l=5%7C%7Cbb=16%7C%7Cbtd=655360%7C%7Cbid=0.64&kp=1&kbc=4cb74286ac23f64b723228f6c9a3296f.d2s&bdrid=4&subBdr=16&kt=240&ki=7215188&ktd=33685820&kbc2=%7C%7Cl=10002%7C%7C%7C%7Crpc=1.88&fdkt=240&fp=WuYBZBUXlhu3xcc5O5yndaTtqF3SFZtg9oPSMgNTV3dFwpyAkVIL0_EY-ffKPK5yPbeHPT3eEofFPx60zdLntu8A5VZAa1Dr8jyRF0pEIKNhiLtCy0fit1i4ZF1fINp5&c=s3cvK_sR__RU9V1pahXKlQ&cme=t19QLxA2Wu6zf35kfoiWLWPT5nS7e0eob5K16CTjI1aW2svvlJCYYMC2iue7FYgPolaEulWQl6RLYC2qdcqX3YhnNA9aGkRQ3QU7yLJQfdE-TGRVGkBSIZ_oE8FkFR8nTkspllFM5et_dW2zjr18ZLHhNthEw8uUUCtz_uJxOaUcr6BtkucpBblGukDUqd_8%7C%7CNDHRnZ9Gz3KXlI-i9OnZqQ==%7C5gDUJdTGiJzedmq9hanWYg==%7CN7fu2vKt8_s=%7CMx272pZD0SF)
* **2.**    [**How to Build a Website**](http://625827376.keywordblocks.com/How_to_Build_a_Website.cfm?&vi=1508243580556658689&dytm=1508243581691&rtbsd=10&npgv=1&kbbq=&sde=1&adepth=1&ddepth=1&tdAdd%5B%5D=%7C@%7Csde=1%7C@%7Cadepth=1%7C@%7Cddepth=1&verid=121199&hvsid=00001508243580966002962362144158&upk=1508243581.2664&sttm=1508243580966&=&vgd_bdata=url_l=4%7C%7Cdevice_l=5%7C%7Cbb=16%7C%7Cbtd=655360%7C%7Cbid=0.64&tdAdd%5B%5D=%7C@%7Cabp:3:2&kp=2&kbc=4cb74286ac23f64b723228f6c9a3296f.d2s&bdrid=4&subBdr=16&kt=240&ki=14336387&ktd=33685820&kbc2=%7C%7Cl=10002%7C%7C%7C%7Crpc=1.87&fdkt=240&fp=WuYBZBUXlhu3xcc5O5yndaTtqF3SFZtg9oPSMgNTV3dFwpyAkVIL0_EY-ffKPK5yPbeHPT3eEofFPx60zdLntu8A5VZAa1Dr8jyRF0pEIKNhiLtCy0fit1i4ZF1fINp5&c=s3cvK_sR__RU9V1pahXKlQ&cme=t19QLxA2Wu6zf35kfoiWLWPT5nS7e0eob5K16CTjI1aW2svvlJCYYMC2iue7FYgPolaEulWQl6RLYC2qdcqX3YhnNA9aGkRQ3QU7yLJQfdE-TGRVGkBSIZ_oE8FkFR8nTkspllFM5et_dW2zjr18ZLHhNthEw8uUUCtz_uJxOaUcr6BtkucpBblGukDUqd_8%7C%7CNDHRnZ9Gz3KXlI-i9OnZqQ==%7C5gDUJdTGiJzedmq9han)
* **3.**    [**Java Certification Courses**](http://625827376.keywordblocks.com/Java_Certification_Courses.cfm?&vi=1508243580556658689&dytm=1508243581691&rtbsd=10&npgv=1&kbbq=&sde=1&adepth=1&ddepth=1&tdAdd%5B%5D=%7C@%7Csde=1%7C@%7Cadepth=1%7C@%7Cddepth=1&verid=121199&hvsid=00001508243580966002962362144158&upk=1508243581.2664&sttm=1508243580966&=&vgd_bdata=url_l=4%7C%7Cdevice_l=5%7C%7Cbb=16%7C%7Cbtd=655360%7C%7Cbid=0.64&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&kp=3&kbc=4cb74286ac23f64b723228f6c9a3296f.d2s&bdrid=4&subBdr=16&kt=240&ki=15713243&ktd=33685820&kbc2=%7C%7Cl=10002%7C%7C%7C%7Crpc=0.92&fdkt=240&fp=WuYBZBUXlhu3xcc5O5yndaTtqF3SFZtg9oPSMgNTV3dFwpyAkVIL0_EY-ffKPK5yPbeHPT3eEofFPx60zdLntu8A5VZAa1Dr8jyRF0pEIKNhiLtCy0fit1i4ZF1fINp5&c=s3cvK_sR__RU9V1pahXKlQ&cme=t19QLxA2Wu6zf35kfoiWLWPT5nS7e0eob5K16CTjI1aW2svvlJCYYMC2iue7FYgPolaEulWQl6RLYC2qdcqX3YhnNA9aGkRQ3QU7yLJQfdE-TGRVGkBSIZ_oE8FkFR8nTkspllFM5et_dW2zjr18ZLHhNthEw8uUUCtz_uJxOaUcr6BtkucpBblGukDUqd_8%7C%7CNDHRnZ9Gz3KXl)
* **4.**    [**Cloud Hosting Services**](http://625827376.keywordblocks.com/Cloud_Hosting_Services.cfm?&vi=1508243580556658689&dytm=1508243581691&rtbsd=10&npgv=1&kbbq=&sde=1&adepth=1&ddepth=1&tdAdd%5B%5D=%7C@%7Csde=1%7C@%7Cadepth=1%7C@%7Cddepth=1&verid=121199&hvsid=00001508243580966002962362144158&upk=1508243581.2664&sttm=1508243580966&=&vgd_bdata=url_l=4%7C%7Cdevice_l=5%7C%7Cbb=16%7C%7Cbtd=655360%7C%7Cbid=0.64&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&kp=4&kbc=4cb74286ac23f64b723228f6c9a3296f.d2s&bdrid=4&subBdr=16&kt=240&ki=272351815&ktd=33685820&kbc2=%7C%7Cl=10002%7C%7C%7C%7Crpc=2.25&fdkt=240&fp=WuYBZBUXlhu3xcc5O5yndaTtqF3SFZtg9oPSMgNTV3dFwpyAkVIL0_EY-ffKPK5yPbeHPT3eEofFPx60zdLntu8A5VZAa1Dr8jyRF0pEIKNhiLtCy0fit1i4ZF1fINp5&c=s3cvK_sR__RU9V1pahXKlQ&cme=t19QLxA2Wu6zf35kfoiWLWPT5nS7e0eob5K16CTjI1aW2svvlJCYYMC2iue7FYgPolaEulWQl6RLYC2qdcqX3YhnNA9aGkRQ3QU7yLJQfdE-TGRVGkBSIZ_oE8FkFR8nTkspllFM5et_dW2zjr18ZLHhNthEw8uUUCtz_uJxOaUcr6BtkucpB)
* **5.**    [**How to Make a Web Page**](http://625827376.keywordblocks.com/How_to_Make_a_Web_Page.cfm?&vi=1508243580556658689&dytm=1508243581691&rtbsd=10&npgv=1&kbbq=&sde=1&adepth=1&ddepth=1&tdAdd%5B%5D=%7C@%7Csde=1%7C@%7Cadepth=1%7C@%7Cddepth=1&verid=121199&hvsid=00001508243580966002962362144158&upk=1508243581.2664&sttm=1508243580966&=&vgd_bdata=url_l=4%7C%7Cdevice_l=5%7C%7Cbb=16%7C%7Cbtd=655360%7C%7Cbid=0.64&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&tdAdd%5B%5D=%7C@%7Cabp:3:2&kp=5&kbc=4cb74286ac23f64b723228f6c9a3296f.d2s&bdrid=4&subBdr=16&kt=240&ki=14358590&ktd=33685820&kbc2=%7C%7Cl=10002%7C%7C%7C%7Crpc=0.21&fdkt=240&fp=WuYBZBUXlhu3xcc5O5yndaTtqF3SFZtg9oPSMgNTV3dFwpyAkVIL0_EY-ffKPK5yPbeHPT3eEofFPx60zdLntu8A5VZAa1Dr8jyRF0pEIKNhiLtCy0fit1i4ZF1fINp5&c=s3cvK_sR__RU9V1pahXKlQ&cme=t19QLxA2Wu6zf35kfoiWLWPT5nS7e0eob5K16CTjI1aW2svvlJCYYMC2iue7FYgPolaEulWQl6RLYC2qdcqX3YhnNA9aGkRQ3QU7yLJQfdE-TGRVGkBSIZ_oE8FkFR8nTkspllFM5et_dW2zjr18Z)

[](http://www.media.net/adchoices?id=625827376)

|  |  |  |
| --- | --- | --- |
| S.NO. | SPLITERATOR | ITERATOR |
| 1. | It is introduced in Java SE 8. | It is available since Java 1.2. |
| 2. | Splitable Iterator | Non-Splitable Iterator |
| 3. | It is used in Stream API. | It is used for Collection API. |
| 4. | It uses Internal Iteration concept to iterate Streams. | It uses External Iteration concept to iterate Collections. |
| 5. | We can use Spliterator to iterate Streams in Parallel and Sequential order. | We can use Iterator to iterate Collections only in Sequential order. |
| 6. | We can get Spliterator by calling spliterator() method on Stream Object. | We can get Iterator by calling iterator() method on Collection Object. |
| 7. | Important Method: tryAdvance() | Important Methods: next(), hasNext() |

**What is Optional in Java 8? What is the use of Optional?Advantages of Java 8 Optional?**

**Optional:**

Optional is a final Class introduced as part of Java SE 8. It is defined in java.util package.

It is used to represent optional values that is either exist or not exist. It can contain either one value or zero value. If it contains a value, we can get it. Otherwise, we get nothing.

It is a bounded collection that is it contains at most one element only. It is an alternative to “null” value.

**Main Advantage of Optional is:**

* It is used to avoid null checks.
* It is used to avoid “NullPointerException”.

**What is Type Inference? Is Type Inference available in older versions like Java 7 and Before 7 or it is available only in Java SE 8?**

Type Inference means determining the Type by compiler at compile-time.

It is not new feature in Java SE 8. It is available in Java 7 and before Java 7 too.

**Before Java 7:-**

Let us explore Java arrays. Define a String of Array with values as shown below:

String str[]={"Java 7","Java 8","Java 9"};

Here we have assigned some String values at right side, but not defined it’s type. Java Compiler automatically infers it’s type and creates a String of Array.

**Java 7:-**

Oracle Corporation has introduced “Diamond Operator” new feature in Java SE 7 to avoid unnecessary Type definition in Generics.

Map<String,List<Customer>> customerInfoByCity =newHashMap<>();

Here we have not defined Type information at right side, simply defined Java SE 7’s Diamond Operator “”.

**Java SE 8:-**

Oracle Corporation has enhanced this Type Inference concept a lot in Java SE 8. We use this concept to define Lambda Expressions, Functions, Method References etc.

ToIntBiFunction<Integer,Integer> add =(a,b)-> a + b;

Here Java Compiler observes the type definition available at left-side and determines the type of Lambda Expression parameters a and b as Integers.

What are the new features introduced in JAVA 8?

There are dozens of features added to Java 8, the most significant ones are mentioned below −

* **Lambda expression** − Adds functional processing capability to Java.
* **Method references** − Referencing functions by their names instead of invoking them directly. Using functions as parameter.
* **Default method** − Interface to have default method implementation.
* **New tools** − New compiler tools and utilities are added like 'jdeps' to figure out dependencies.
* **Stream API** − New stream API to facilitate pipeline processing.
* **Date Time API** − Improved date time API.
* **Optional** − Emphasis on best practices to handle null values properly.
* **Nashorn, JavaScript Engine** − A Java-based engine to execute JavaScript code.

Along with these new featuers, lots of feature enhancements are done under-the-hood, at both compiler and JVM level.

How will you sort a list of string using Java 8 lambda expression?

Following code sorts a list of string using Java 8 lambda expression:

//sort using java 8privatevoid sortUsingJava8(List<String> names){Collections.sort(names,(s1, s2)-> s1.compareTo(s2));}

What are the characteristics of a Java 8 lambda expression?

A lambda expression is characterized by the following syntax -

parameter −> expression body

Following are the important characteristics of a lambda expression −

* **Optional type declaration** − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
* **Optional parenthesis around parameter** − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
* **Optional curly braces** − No need to use curly braces in expression body if the body contains a single statement.
* **Optional return keyword** − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Why lambda expression is to be used?

Lambda expressions are used primarily to define inline implementation of a functional interface, i.e., an interface with a single method only. In the above example, we've used various types of lambda expressions to define the operation method of MathOperation interface. Then we have defined the implementation of sayMessage of GreetingService.

Lambda expression eliminates the need of anonymous class and gives a very simple yet powerful functional programming capability to Java.

What kind of variable you can access in an lambda expression??

Using lambda expression, you can refer to final variable or effectively final variable (which is assigned only once). Lambda expression throws a compilation error, if a variable is assigned a value the second time.

What are method references?

Method references help to point to methods by their names. A method reference is described using :: (double colon) symbol. A method reference can be used to point the following types of methods −

* Static methods
* Instance methods
* Constructors using new operator (TreeSet::new)

Explain the System.out::println expression.

System.out::println method is a static method reference to println method of out object of System class.

What are functional interfaces?

Functional interfaces have a single functionality to exhibit. For example, a Comparable interface with a single method 'compareTo' is used for comparison purpose. Java 8 has defined a lot of functional interfaces to be used extensively in lambda expressions.

What is the purpose of BiConsumer<T,U> functional interface?

It represents an operation that accepts two input arguments, and returns no result.

What is the purpose of BiFunction<T,U,R> functional interface?

It represents a function that accepts two arguments and produces a result.

What is the purpose of BinaryOperator<T> functional interface?

It represents an operation upon two operands of the same type, producing a result of the same type as the operands.

What is the purpose of BiPredicate<T,U> functional interface?

It represents a predicate (Boolean-valued function) of two arguments.

What is the purpose of BooleanSupplier functional interface?

It represents a supplier of Boolean-valued results.

What is the purpose of Consumer<T> functional interface?

It represents an operation that accepts a single input argument and returns no result.

What is the purpose of DoubleBinaryOperator functional interface?

It represents an operation upon two double-valued operands and producing a double-valued result.

What is the purpose of DoubleConsumer functional interface?

It represents an operation that accepts a single double-valued argument and returns no result.

What is the purpose of DoubleFunction<R> functional interface?

It represents a function that accepts a double-valued argument and produces a result.

What is the purpose of DoublePredicate functional interface?

It represents a predicate (Boolean-valued function) of one double-valued argument.

What is the purpose of DoubleSupplier functional interface?

It represents a supplier of double-valued results.

What is the purpose of DoubleToIntFunction functional interface?

It represents a function that accepts a double-valued argument and produces an int-valued result.

What is the purpose of DoubleToLongFunction functional interface?

It represents a function that accepts a double-valued argument and produces a long-valued result.

What is the purpose of DoubleUnaryOperator functional interface?

It represents an operation on a single double-valued operand that produces a double-valued result.

What is the purpose of Function<T,R> functional interface?

It represents a function that accepts one argument and produces a result.

What is the purpose of IntBinaryOperator functional interface?

It represents an operation upon two int-valued operands and produces an int-valued result.

What is the purpose of IntConsumer functional interface?

It represents an operation that accepts a single int-valued argument and returns no result.

What is the purpose of IntFunction<R> functional interface?

It represents a function that accepts an int-valued argument and produces a result.

What is the purpose of IntPredicate functional interface?

It represents a predicate (Boolean-valued function) of one int-valued argument.

What is the purpose of IntSupplier functional interface?

It represents a supplier of int-valued results.

What is the purpose of IntToDoubleFunction functional interface?

It represents a function that accepts an int-valued argument and produces a double-valued result.

What is the purpose of IntToLongFunction functional interface?

It represents a function that accepts an int-valued argument and produces a long-valued result.

What is the purpose of IntUnaryOperator functional interface?

It represents an operation on a single int-valued operand that produces an int-valued result.

What is the purpose of LongBinaryOperator functional interface?

It represents an operation upon two long-valued operands and produces a long-valued result.

What is the purpose of LongConsumer functional interface?

It represents an operation that accepts a single long-valued argument and returns no result.

What is the purpose of LongFunction<R> functional interface?

It represents a function that accepts a long-valued argument and produces a result.

What is the purpose of LongPredicate functional interface?

It represents a predicate (Boolean-valued function) of one long-valued argument.

What is the purpose of LongSupplier functional interface?

It represents a supplier of long-valued results.

What is the purpose of LongToDoubleFunction functional interface?

It represents a function that accepts a long-valued argument and produces a double-valued result.

What is the purpose of LongToIntFunction functional interface?

It represents a function that accepts a long-valued argument and produces an int-valued result.

What is the purpose of LongUnaryOperator functional interface?

It represents an operation on a single long-valued operand that produces a long-valued result.

What is the purpose of ObjDoubleConsumer<T> functional interface?

It represents an operation that accepts an object-valued and a double-valued argument, and returns no result.

What is the purpose of ObjIntConsumer<T> functional interface?

It represents an operation that accepts an object-valued and an int-valued argument, and returns no result.

What is the purpose of ObjLongConsumer<T> functional interface?

It represents an operation that accepts an object-valued and a long-valued argument, and returns no result.

What is the purpose of Predicate<T> functional interface?

It represents a predicate (Boolean-valued function) of one argument.

What is the purpose of Supplier<T> functional interface?

It represents a supplier of results.

What is the purpose of ToDoubleBiFunction<T,U> functional interface?

It represents a function that accepts two arguments and produces a double-valued result.

What is the purpose of ToDoubleFunction<T> functional interface?

It represents a function that produces a double-valued result.

What is the purpose of ToIntBiFunction<T,U> functional interface?

It represents a function that accepts two arguments and produces an int-valued result.

What is the purpose of ToIntFunction<T> functional interface?

It represents a function that produces an int-valued result.

What is the purpose of ToLongBiFunction<T,U> functional interface?

It represents a function that accepts two arguments and produces a long-valued result.

What is the purpose of ToLongFunction<T> functional interface?

It represents a function that produces a long-valued result.

What is the purpose of UnaryOperator<T> functional interface?

It represents an operation on a single operand that produces a result of the same type as its operand.

What are default methods?

With java 8, an interface can have default implementation of a function in interfaces.

What are static default methods?

An interface can also have static helper methods from Java 8 onwards.

publicinterface vehicle {defaultvoidprint(){System.out.println("I am a vehicle!");}staticvoid blowHorn(){System.out.println("Blowing horn!!!");}}

How will you call a default method of an interface in a class?

Using super keyword along with interface name.

interfaceVehicle{defaultvoidprint(){System.out.println("I am a vehicle!");}}classCarimplementsVehicle{publicvoidprint(){Vehicle.super.print();}}

How will you call a static method of an interface in a class?

Using name of the interface.

interfaceVehicle{staticvoid blowHorn(){System.out.println("Blowing horn!!!");}}classCarimplementsVehicle{publicvoidprint(){Vehicle.blowHorn();}}

What is streams in Java 8?

Stream represents a sequence of objects from a source, which supports aggregate operations.

What is stream pipelining in Java 8?

Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.

What is the difference between Collections and Stream in Java8 ?

Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

What is the purpose of forEach method of stream in java 8?

Stream has provided a new method 'forEach' to iterate each element of the stream.

How will you print 10 random numbers using forEach of java 8?

The following code segment shows how to print 10 random numbers using forEach.

Random random =newRandom(); random.ints().limit(10).forEach(System.out::println);

What is the purpose of map method of stream in java 8?

The 'map' method is used to map each element to its corresponding result.

How will you print unique squares of numbers in java 8?

The following code segment prints unique squares of numbers using map.

List<Integer> numbers =Arrays.asList(3,2,2,3,7,3,5);//get list of unique squaresList<Integer> squaresList = numbers.stream().map( i -> i\*i).distinct().collect(Collectors.toList());

What is the purpose of filter method of stream in java 8?

The 'filter' method is used to eliminate elements based on a criteria.

How will you print count of empty strings in java 8?

The following code segment prints a count of empty strings using filter.

List<String>strings =Arrays.asList("abc","","bc","efg","abcd","","jkl");//get count of empty stringint count = strings.stream().filter(string−>string.isEmpty()).count();

What is the purpose of limit method of stream in java 8?

The 'limit' method is used to reduce the size of the stream.

How will you print 10 random numbers in java 8?

The following code segment shows how to print 10 random numbers.

Random random =newRandom(); random.ints().limit(10).forEach(System.out::println);

What is the purpose of sorted method of stream in java 8?

The 'sorted' method is used to sort the stream.

How will you print 10 random numbers in a sorted order in java 8?

The following code segment shows how to print 10 random numbers in a sorted order.

Random random =newRandom(); random.ints().limit(10).sorted().forEach(System.out::println);

What is Parallel Processing in Java 8?

parallelStream is the alternative of stream for parallel processing. Take a look at the following code segment that prints a count of empty strings using parallelStream.

List<String> strings =Arrays.asList("abc","","bc","efg","abcd","","jkl");//get count of empty stringint count = strings.parallelStream().filter(string−>string.isEmpty()).count();//It is very easy to switch between sequential and parallel streams.

What are collectors in Java 8?

Collectors are used to combine the result of processing on the elements of a stream. Collectors can be used to return a list or a string.

List<String>strings =Arrays.asList("abc","","bc","efg","abcd","","jkl");List<String> filtered = strings.stream().filter(string->!string.isEmpty()).collect(Collectors.toList());System.out.println("Filtered List: "+ filtered);String mergedString = strings.stream().filter(string->!string.isEmpty()).collect(Collectors.joining(", "));System.out.println("Merged String: "+ mergedString);

What are Statistics collectors in Java 8?

With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.

How will you get the highest number present in a list using Java 8?

Following code will print the highest number present in a list.

List<Integer> numbers =Arrays.asList(3,2,2,3,7,3,5);IntSummaryStatistics stats = integers.stream().mapToInt((x)−> x).summaryStatistics();System.out.println("Highest number in List : "+ stats.getMax());

How will you get the lowest number present in a list using Java 8?

Following code will print the highest number present in a list.

List<Integer> numbers =Arrays.asList(3,2,2,3,7,3,5);IntSummaryStatistics stats = integers.stream().mapToInt((x)−> x).summaryStatistics();System.out.println("Lowest number in List : "+ stats.getMin());

How will you get the sum of all numbers present in a list using Java 8?

Following code will print the sum of all numbers present in a list.

List<Integer> numbers =Arrays.asList(3,2,2,3,7,3,5);IntSummaryStatistics stats = integers.stream().mapToInt((x)−> x).summaryStatistics();System.out.println("Sum of all numbers : "+ stats.getSum());

How will you get the average of all numbers present in a list using Java 8?

Following code will print the average of all numbers present in a list.

List<Integer> numbers =Arrays.asList(3,2,2,3,7,3,5);IntSummaryStatistics stats = integers.stream().mapToInt((x)−> x).summaryStatistics();System.out.println("Average of all numbers : "+ stats.getAverage());

What is Optional in Java8?

Optional is a container object which is used to contain not-null objects. Optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as 'available' or 'not available' instead of checking null values. It is introduced in Java 8 and is similar to what Optional is in Guava.

What is Nashorn in Java8?

With Java 8, Nashorn, a much improved javascript engine is introduced, to replace the existing Rhino. Nashorn provides 2 to 10 times better performance, as it directly compiles the code in memory and passes the bytecode to JVM. Nashorn uses invokedynamics feature, introduced in Java 7 to improve performance.

What is jjs in JAVA8?

For Nashorn engine, JAVA 8 introduces a new command line tool, jjs, to execute javascript codes at console.

Can you execute javascript code from java 8 code base?

Yes! Using ScriptEngineManager, JavaScript code can be called and interpreted in Java.

What is local datetime API in JAVA8?

Local − Simplified date-time API with no complexity of timezone handling.

What is zoned datetime API in JAVA8?

Zoned − Specialized date-time API to deal with various timezones.

What is chromounits in java8?

java.time.temporal.ChronoUnit enum is added in Java 8 to replace the integer values used in old API to represent day, month, etc.

How will you get the current date using local datetime api of java8?

Following code gets the current date using local datetime api −

//Get the current dateLocalDate today =LocalDate.now();System.out.println("Current date: "+ today);

How will you add 1 week to current date using local datetime api of java8?

Following code adds 1 week to current date using local datetime api −

//add 1 week to the current dateLocalDate today =LocalDate.now();LocalDate nextWeek = today.plus(1,ChronoUnit.WEEKS);System.out.println("Next week: "+ nextWeek);

How will you add 1 month to current date using local datetime api of java8?

Following code adds 1 month to current date using local datetime api:

//add 1 month to the current dateLocalDate today =LocalDate.now();LocalDate nextMonth = today.plus(1,ChronoUnit.MONTHS);System.out.println("Next month: "+ nextMonth);

How will you add 1 year to current date using local datetime api of java8?

Following code adds 1 year to current date using local datetime api −

//add 1 year to the current dateLocalDate today =LocalDate.now();LocalDate nextYear = today.plus(1,ChronoUnit.YEARS);System.out.println("Next year: "+ nextYear);

How will you add 10 years to current date using local datetime api of java8?

Following code adds 10 years to current date using local datetime api −

//add 10 years to the current dateLocalDate today =LocalDate.now();LocalDate nextDecade = today.plus(1,ChronoUnit.DECADES);System.out.println("Date after ten year: "+ nextDecade);

How will you get next tuesday using java8?

Following code gets next tuesday using java8 −

//get the next tuesdayLocalDate today =LocalDate.now();LocalDate nextTuesday = today.with(TemporalAdjusters.next(DayOfWeek.TUESDAY));System.out.println("Next Tuesday on : "+ nextTuesday);

How will you get second saturday of next month using java8?

Following code gets second saturday of next month using java8 −

//get the second saturday of next monthLocalDate firstInYear =LocalDate.of(date1.getYear(),date1.getMonth(),1);LocalDate secondSaturday = firstInYear.with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY)).with(TemporalAdjusters.next(DayOfWeek.SATURDAY));System.out.println("Second Saturday on : "+ secondSaturday);

How will you get the instant of current date in terms of milliseconds using java8?

Following code gets the instant of current date in terms of milliseconds −

//Get the instant of current date in terms of millisecondsInstant now = currentDate.toInstant();

How will you get the instant of local date time using time in of milliseconds using java8?

Following code gets the instant of local date time using time in of milliseconds −

Instant now = currentDate.toInstant();ZoneId currentZone =ZoneId.systemDefault();LocalDateTime localDateTime =LocalDateTime.ofInstant(now, currentZone);System.out.println("Local date: "+ localDateTime);

How will you get the instant of zoned date time using time in of milliseconds using java8?

Following code gets the instant of zoned date time using time in of milliseconds −

Instant now = currentDate.toInstant();ZoneId currentZone =ZoneId.systemDefault();ZonedDateTime zonedDateTime =ZonedDateTime.ofInstant(now, currentZone);System.out.println("Zoned date: "+ zonedDateTime);

Which class implements a decoder for decoding byte data using the Base64 encoding scheme in Java8?

static class Base64.Decoder − This class implements a decoder for decoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045.

Which class implements an encoder for encoding byte data using the Base64 encoding scheme in Java8?

static class Base64.Encoder − This class implements an encoder for encoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045.

How will you create a Base64 decoder?

getDecoder() method of Base64 class returns a Base64.Decoder that decodes using the Basic type base64 encoding scheme.

How will you create a Base64 encoder?

getEncoder() method of Base64 class returns a Base64.Encoder that encodes using the Basic type base64 encoding scheme.

How will you create a Base64 decoder that decodes using the MIME type base64 encoding scheme?

getMimeDecoder() method of Base64 class returns a Base64.Decoder that decodes using the MIME type base64 decoding scheme.

How will you create a Base64 encoder that encodes using the MIME type base64 encoding scheme?

getMimeEncoder() method of Base64 class returns a Base64.Encoder that encodes using the MIME type base64 encoding scheme.

How will you create a Base64 decoder that decodes using the URL and Filename safe type base64 encoding scheme?

getUrlDecoder() method of Base64 class returns a Base64.Decoder that decodes using the URL and Filename safe type base64 encoding scheme.

How will you create a Base64 encoder that encodes using the URL and Filename safe type base64 encoding scheme?

getUrlEncoder() method of Base64 class returns a Base64.Encoder that encodes using the URL and Filename safe type base64 encoding scheme.