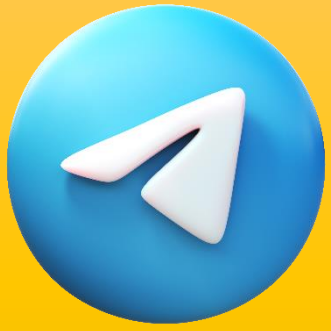
Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group



Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Java 8 Interview Questions and Answers

Q. What are the important features of Java 8 release?

- Interface methods by default;
- Lambda expressions;
- Functional interfaces;
- References to methods and constructors;
- Repeatable annotations
- Annotations on data types;
- Reflection for method parameters;
- Stream API for working with collections;
- Parallel sorting of arrays;
- New API for working with dates and times;
- New JavaScript Nashorn Engine ;
- Added several new classes for thread safe operation;
- Added a new API for `Calendar`and `Locale`;
- Added support for Unicode 6.2.0 ;
- Added a standard class for working with Base64 ;
- Added support for unsigned arithmetic;
- Improved constructor `java.lang.String(byte[], *)` and method performance `java.lang.String.getBytes()`;
- A new implementation `AccessController.doPrivileged` that allows you to set a subset of privileges without having to check all * other access levels;
- Password-based algorithms have become more robust;
- Added support for SSL / TLS Server Name Indication (NSI) in JSSE Server ;
- Improved keystore (KeyStore);
- Added SHA-224 algorithm;
- Removed JDBC Bridge - ODBC;
- PermGen is removed , the method for storing meta-data of classes is changed;
- Ability to create profiles for the Java SE platform, which include not the entire platform, but some part of it;
- Tools
  - Added utility `jjs` for using JavaScript Nashorn;
  - The command `java` can run JavaFX applications;
  - Added utility `jdeps` for analyzing .class files.

`<b><a href="#">↥ back to top</a></b>`

Q. Can you declare an interface method static?

Java 8 interface changes include static methods and default methods in interfaces. Prior to Java 8, we could have only method declarations in the interfaces. But from Java 8, we can have default methods and static methods in the interfaces.

`<b><a href="#">↥ back to top</a></b>`

Q. What is a lambda?

What is the structure and features of using a lambda expression? A lambda is a set of instructions that can be separated into a separate variable and then repeatedly called in various places of the program.

The basis of the lambda expression is the *lambda operator* , which represents the arrow - >. This operator divides the lambda expression into two parts: the left side contains a list of expression parameters, and the right actually represents the body of the lambda expression, where all actions are performed.

The lambda expression is not executed by itself, but forms the implementation of the method defined in the functional interface. It is important that the functional interface should contain only one single method without implementation.

```
interface  Operationable {
     int  calculate ( int  x , int  y );
}

public  static  void main ( String [] args) {
    Operationable operation = (x, y) - > x + y;
    int result = operation.calculate ( 10 , 20 );
    System.out.println (result); // 30
}
```

In fact, lambda expressions are in some way a shorthand form of internal anonymous classes that were previously used in Java.

- *Deferred execution lambda expressions* - it is defined once in one place of the program, it is called if necessary, any number of times and in any place of the program.
- *The parameters of the lambda expression* must correspond in type to the parameters of the functional interface method:

```
operation = ( int x, int y) - > x + y;
// When writing the lambda expression itself, the parameter type is allowe
(x, y) - > x + y;
// If the method does not accept any parameters, then empty brackets are w
() - >  30  +  20 ;
// If the method accepts only one parameter, then the brackets can be omit
n - > n * n;
```

- Trailing lambda expressions are not required to return any value.

```
interface  Printable {
     void  print( String  s );
}

public  static  void main ( String [] args) {
    Printable printer = s - >  System.out.println(s);
    printer.print("Hello, world");
}

// _ Block lambda - expressions_ are surrounded by curly braces . The modu


Operationable operation = ( int x, int y) - > {
    if (y ==  0 ) {
        return  0 ;
    }
    else {
        return x / y;
    }
};
```

- Passing a lambda expression as a method parameter

```
interface  Condition {
     boolean  isAppropriate ( int  n );
}

private  static  int sum ( int [] numbers, Condition condition) {
     int result =  0 ;
    for ( int i : numbers) {
         if (condition.isAppropriate(i)) {
            result + = i;
         }
    }
    return result;
}

public  static  void main ( String [] args) {
     System.out.println(sum ( new  int [] { 0 , 1 , 0 , 3 , 0 , 5 , 0 , 7
}
```

↑ back to top

Q. What variables do lambda expressions have access to?

Access to external scope variables from a lambda expression is very similar to access from anonymous objects.

- immutable (effectively final - not necessarily marked as final) local variables;
- class fields
- static variables.

The default methods of the implemented functional interface are not allowed to be accessed inside the lambda expression.

↑ back to top

Q. How to sort a list of strings using a lambda expression?

```
public  static  List < String > sort ( List < String > list) {
    Collections.sort(list, (a, b) -> a.compareTo(b));
    return list;
}
```

↑ back to top

Q. What is a method reference?

If the method existing in the class already does everything that is necessary, then you can use the method reference mechanism (method reference) to directly pass this method. The result will be exactly the same as in the case of defining a lambda expression that calls this method.

```
private  interface  Measurable {
    public  int  length ( String  string );
}

public  static  void main ( String [] args) {
    Measurable a =  String::length;
    System.out.println(a.length("abc"));
}
```

Method references are potentially more efficient than using lambda expressions. In

addition, they provide the compiler with better information about the type, and if you can choose between using a reference to an existing method and using a lambda expression, you should always use a method reference.

↥ back to top

Q. What types of method references do you know?

- on the static method;
- per instance method;
- to the constructor.

↥ back to top

Q. Explain the expression `System.out::println`?

The specified expression illustrates passing a reference to a static method of a `println()`class `System.out`.

↥ back to top

Q. What is a Functional Interface?

A **functional interface** is an interface that defines only one abstract method.

To accurately determine the interface as functional, an annotation has been added `@FunctionalInterface` that works on the principle of `@Override`. It will designate a plan and will not allow to define the second abstract method in the interface.

An interface can include as many `default` methods as you like while remaining functional, because `default` methods are not abstract.

↥ back to top

Q. What is StringJoiner?

The class is StringJoinerused to create a sequence of strings separated by a separator with the ability to append a prefix and suffix to the resulting string:

```
StringJoiner joiner =  new  StringJoiner ( " . " , " Prefix- " , " -suffix
for ( String s :  " Hello the brave world " . split ( "  " )) {
    , joiner, . add (s);
}
System.out.println(joiner); // prefix-Hello.the.brave.world-suffix
```

↥ back to top

Q. What are `default`interface methods?

Java 8 allows you to add non-abstract method implementations to an interface using the keyword default:

```
interface  Example {
    int  process ( int  a );
    default void  show () {
        System.out.println("default show ()");
    }
}
```

- If a class implements an interface, it can, but does not have to, implement the default

methods already implemented in the * interface. The class inherits the default implementation.
- If a class implements several interfaces that have the same default method, then the class must implement the method with the same signature on its own. The situation is similar if one interface has a default method, and in the other the same method is abstract - no class default implementation is inherited.
- The default method cannot override the class method `java.lang.Object`.
- They help implement interfaces without fear of disrupting other classes.
- Avoid creating utility classes, since all the necessary methods can be represented in the interfaces themselves.
- They give classes the freedom to choose the method to be redefined.
- One of the main reasons for introducing default methods is the ability of collections in Java 8 to use lambda expressions.


```
<b><a href="#">↥ back to top</a></b>
```


Q. How to call `default` interface method in a class that implements this interface?

Using the keyword superalong with the interface name:

```
interface  Paper {
    default void  show () {
        System.out.println(" default show ()");
    }
}

class  License  implements  Paper {
     public  void  show () {
        Paper.super.show();
    }
}
```


```
<b><a href="#">↥ back to top</a></b>
```


Q. What is `static` interface method?

Static interface methods are similar to default methods, except that there is no way to override them in classes that implement the interface.

- Static methods in the interface are part of the interface without the ability to use them for objects of the implementation class
- Class methods `java.lang.Object`cannot be overridden as static
- Static methods in the interface are used to provide helper methods, for example, checking for null, sorting collections, etc.


```
<b><a href="#">↥ back to top</a></b>
```


Q. How to call `static` interface method?

Using the interface name:

```
interface  Paper {
     static  void  show () {
        System.out.println( " static show () " );
    }
}

class  License {
     public  void  showPaper () {
        Paper.show ();
    }
}
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is Optional

An optional value `Optional`is a container for an object that may or may not contain a value `null`. Such a wrapper is a convenient means of prevention `NullPointerException`, as has some higher-order functions, eliminating the need for repeating `if null/notNull`checks:

```
Optional < String > optional =  Optional . of ( " hello " );

optional.isPresent(); // true
optional.ifPresent(s ->  System.out.println(s . length ())); // 5
optional.get(); // "hello"
optional.orElse( " ops ... " ); // "hello"

<b><a href="#">↥ back to top</a></b>
```

Q. What is Stream?

An interface `java.util.Stream` is a sequence of elements on which various operations can be performed.

Operations on streams can be either intermediate (intermediate) or final (terminal) . Final operations return a result of a certain type, and intermediate operations return the same stream. Thus, you can build chains of several operations on the same stream.

A stream can have any number of calls to intermediate operations and the last call to the final operation. At the same time, all intermediate operations are performed lazily and until the final operation is called, no actions actually happen (similar to creating an object `Thread`or `Runnable`, without a call `start()`).

Streams are created based on sources of some, for example, classes from `java.util.Collection`.

Associative arrays (maps), for example `HashMap`, are not supported.

Operations on streams can be performed both sequentially and in parallel.

Streams cannot be reused. As soon as some final operation has been called, the flow is closed.

In addition to the universal object, there are special types of streams to work with primitive data types `int`, `long`and `double`: `IntStream`, `LongStream`and `DoubleStream`. These primitive streams work just like regular object streams, but with the following differences:

- use specialized lambda expressions, for example, `IntFunction`or `IntPredicate`instead of `Function`and `Predicate`;
- support additional end operations `sum()`, `average()`, `mapToObj()`.

```
<b><a href="#">↥ back to top</a></b>
```

Q. What are the ways to create a stream?

- Using collection:

```
Stream < String > fromCollection =  Arrays.asList ( " x " , " y " , " z "
```

- Using set of values:

```
Stream < String > fromValues =  Stream.of( " x " , " y " , " z " );
```

- Using Array

```
Stream < String > fromArray =  Arrays.stream( new  String [] { " x " , " y
```

- Using file (each line in the file will be a separate element in the stream):

```
Stream < String > fromFile =  Files.lines( Paths.get(" input.txt "));
```

- From the line:

```
IntStream fromString =  " 0123456789 " . chars ();
```

- With the help of `Stream.builder()`:

```
Stream < String > fromBuilder =  Stream.builder().add (" z ").add(" y ").a
```

- Using `Stream.iterate()(infinite)`:

```
Stream < Integer > fromIterate =  Stream.iterate ( 1 , n - > n +  1 );
```

- Using `Stream.generate()(infinite)`:

```
Stream < String > fromGenerate =  Stream.generate(() ->  " 0 " );
```

↥ back to top

Q. What is the difference between `Collection` and `Stream`?

Collections allow you to work with elements separately, while streams do not allow this, but instead provides the ability to perform functions on data as one.

↥ back to top

Q. What is the method `collect()`for streams for?

A method `collect()`is the final operation that is used to represent the result as a collection or some other data structure.

`collect()`accepts an input that contains four stages:

- **supplier** — initialization of the battery,
- **accumulator** — processing of each element,
- **combiner** — connection of two accumulators in parallel execution,
- **[finisher]** —a non-mandatory method of the last processing of the accumulator.

In Java 8, the class `Collectors` implements several common collectors:

- `toList()`, `toCollection()`, `toSet()`- present stream in the form of a list, collection or set;
- `toConcurrentMap()`, `toMap()`- allow you to convert the stream to `Map`;
- `averagingInt()`, `averagingDouble()`, `averagingLong()`- return the average value;
- `summingInt()`, `summingDouble()`, `summingLong()`- returns the sum;
- `summarizingInt()`, `summarizingDouble()`, `summarizingLong()`- return SummaryStatisticswith different values of the aggregate;
- `partitioningBy()`- divides the collection into two parts according to the condition and returns them as `Map<Boolean, List>`;
- `groupingBy()`- divides the collection into several parts and returns `Map<N, List<T>>`;
- `mapping()`- Additional value conversions for complex Collectors.

There is also the possibility of creating your own collector through `Collector.of()`:

```
Collector < String , a List < String > , a List < String > > toList =  Col
    ArrayList :: new ,
    List :: add,
    (l1, l2) -> {l1 . addAll (l2); return l1; }
);
```

↥ back to top

Q. Why do streams use `forEach()`and `forEachOrdered()` methods?

- `forEach()` applies a function to each stream object; ordering in parallel execution is not guaranteed;
- `forEachOrdered()` applies a function to each stream object while maintaining the order of the elements.

↥ back to top

Q. What are `map()`, `mapToInt()`, `mapToDouble()` and `mapToLong()` methods in Stream?

The method `map()`is an intermediate operation, which transforms each element of the stream in a specified way.

`mapToInt()`, `mapToDouble()`, `mapToLong()`- analogues `map()`, returns the corresponding numerical stream (ie the stream of numerical primitives):

```
Stream
    .of ( " 12 " , " 22 " , " 4 " , " 444 " , " 123 " )
    .mapToInt ( Integer :: parseInt)
    .toArray (); // [12, 22, 4, 444, 123]
```

↥ back to top

Q. What is the purpose of `filter()` method in streams?

The method `filter()` is an intermediate operation receiving a predicate that filters all elements, returning only those that match the condition.

↥ back to top

Q. What is the use of `limit()` method in streams?

The method `limit()`is an intermediate operation, which allows you to limit the selection to a certain number of first elements.

↥ back to top

Q. What is the use of `sorted()` method in streams?

The method `sorted()`is an intermediate operation, which allows you to sort the values either in natural order or by setting Comparator.

The order of the elements in the original collection remains untouched - `sorted()`it just creates its sorted representation.

↥ back to top

Q. What streamers designed methods `flatMap()`, `flatMapToInt()`, `flatMapToDouble()`, `flatMapToLong()`?

The method `flatMap()` is similar to map, but can create several from one element. Thus, each object will be converted to zero, one or more other objects supported by the stream. The most obvious way to use this operation is to convert container elements using functions that return containers.

```
Stream
    .of ( " Hello " , " world! " )
    .flatMap ((p) ->  Arrays.stream (p . split ( " , " )))
    .toArray ( String [] :: new ); // ["H", "e", "l", "l", "o", "w", "o",
```

`flatMapToInt()`, `flatMapToDouble()`, `flatMapToLong()`- are analogues `flatMap()`, returns the corresponding numerical stream.

↥ back to top

Q. Tell us about parallel processing in Java 8?

Streams can be sequential and parallel. Operations on sequential streams are performed in one processor thread, on parallel streams - using several processor threads. Parallel streams use the shared stream `ForkJoinPool`through the static `ForkJoinPool.commonPool()`method. In this case, if the environment is not multi-core, then the stream will be executed as sequential. In fact, the use of parallel streams is reduced to the fact that the data in the streams will be divided into parts, each part is processed on a separate processor core, and in the end these parts are connected, and final operations are performed on them.

You can also use the `parallelStream()`interface method to create a parallel stream from the collection `Collection`.

To make a regular sequential stream parallel, you must call the `Stream`method on the object `parallel()`. The method `isParallel()`allows you to find out if the stream is parallel.

Using, methods `parallel()`and `sequential()`it is possible to determine which operations can be parallel, and which only sequential. You can also make a parallel stream from any sequential stream and vice versa:

```
collection
  .stream ()
  .peek ( ... ) // operation is sequential
  .parallel ()
  .map ( ... ) // the operation can be performed in parallel,
  .sequential ()
  .reduce ( ... ) // operation is sequential again
```

As a rule, elements are transferred to the stream in the same order in which they are defined in the data source. When working with parallel streams, the system preserves the sequence of elements. An exception is a method `forEach()`that can output elements in random order. And in order to maintain the order, it is necessary to apply the method `forEachOrdered()`.

- Criteria that may affect performance in parallel streams:
- Data size - the more data, the more difficult it is to separate the data first, and then combine them.
- The number of processor cores. Theoretically, the more cores in a computer, the faster the program will work. If the machine has one core, it makes no sense to use parallel threads.
- The simpler the data structure the stream works with, the faster operations will occur. For example, data from is `ArrayList`easy to use, since the structure of this collection assumes a sequence of unrelated data. But a type collection `LinkedList`is not the best option, since in a sequential list all the elements are connected with previous / next. And such data is difficult to parallelize.

- Operations with primitive types will be faster than with class objects.
- It is highly recommended that you do not use parallel streams for any long operations (for example, network connections), since all parallel streams work with one `ForkJoinPool`, such long operations can stop all parallel streams in the JVM due to the lack of available threads in the pool, etc. e. parallel streams should be used only for short operations where the count goes for milliseconds, but not for those where the count can go for seconds and minutes;
- Saving order in parallel streams increases execution costs, and if order is not important, it is possible to disable its saving and thereby increase productivity by using an intermediate operation `unordered()`:

```
collection.parallelStream ()
    .sorted ()
    .unordered ()
    .collect ( Collectors . toList ());
```

↥ back to top

Q. What are the final methods of working with streams you know?

- `findFirst()` returns the first element
- `findAny()` returns any suitable item
- `collect()` presentation of results in the form of collections and other data structures
- `count()` returns the number of elements
- `anyMatch()`returns trueif the condition is satisfied for at least one element
- `noneMatch()`returns trueif the condition is not satisfied for any element
- `allMatch()`returns trueif the condition is satisfied for all elements
- `min()`returns the minimum element, using as a condition Comparator
- `max()`returns the maximum element, using as a condition Comparator
- `forEach()` applies a function to each object (order is not guaranteed in parallel execution)
- `forEachOrdered()` applies a function to each object while preserving the order of elements
- `toArray()` returns an array of values
- `reduce()`allows you to perform aggregate functions and return a single result.
- `sum()` returns the sum of all numbers
- `average()` returns the arithmetic mean of all numbers.

↥ back to top

Q. What intermediate methods of working with streams do you know?

- `filter()` filters records, returning only records matching the condition;
- `skip()` allows you to skip a certain number of elements at the beginning;
- `distinct()`returns a stream without duplicates (for a method `equals()`);
- `map()` converts each element;
- `peek()` returns the same stream, applying a function to each element;
- `limit()` allows you to limit the selection to a certain number of first elements;
- `sorted()`allows you to sort values either in natural order or by setting `Comparator`;
- `mapToInt()`, `mapToDouble()`, `mapToLong()`- analogues `map()`return stream numeric primitives;
- `flatMap()`, `flatMapToInt()`, `flatMapToDouble()`, `flatMapToLong()`- similar to `map()`, but can create a single element more.

For numerical streams, an additional method is available `mapToObj()`that converts the numerical stream back to the object stream.

↥ back to top

Q. How to display 10 random numbers using forEach()?

```
( new  Random ())
    .ints ()
    .limit ( 10 )
    .forEach ( System . out :: println);

<b><a href="#">↥ back to top</a></b>
```

Q. How can I display unique squares of numbers using the method map()?

```
Stream
    .of ( 1 , 2 , 3 , 2 , 1 )
    .map (s -> s * s)
    .distinct ()
    .collect ( Collectors . toList ())
    .forEach ( System . out :: println);

<b><a href="#">↥ back to top</a></b>
```

Q. How to display the number of empty lines using the method filter()?

```
System.out.println (
     Stream
        .of ( " Hello " , " " , " , " , " world " , " ! " )
        .filter ( String :: isEmpty)
        .count ());

<b><a href="#">↥ back to top</a></b>
```

Q. How to display 10 random numbers in ascending order?

```
( new  Random ())
    .ints ()
    .limit ( 10 )
    .sorted ()
    .forEach ( System . out :: println);

<b><a href="#">↥ back to top</a></b>
```

Q. How to find the maximum number in a set?

```
Stream
    .of ( 5 , 3 , 4 , 55 , 2 )
    .mapToInt (a -> a)
    .max ()
    .getAsInt (); // 55

<b><a href="#">↥ back to top</a></b>
```

Q. How to find the minimum number in a set?

```
Stream
    .of ( 5 , 3 , 4 , 55 , 2 )
    .mapToInt (a -> a)
    .min ()
    .getAsInt (); // 2
```

↑ back to top

Q. How to get the sum of all numbers in a set?

```
Stream
    .of( 5 , 3 , 4 , 55 , 2 )
    .mapToInt()
    .sum(); // 69
```

↑ back to top

Q. How to get the average of all numbers?

```
Stream
    .of ( 5 , 3 , 4 , 55 , 2 )
    .mapToInt (a -> a)
    .average ()
    .getAsDouble (); // 13.8
```

Q. What additional methods for working with associative arrays (maps) appeared in Java 8?

- `putIfAbsent()` adds a key-value pair only if the key was missing:

```
map.putIfAbsent("a", "Aa");
```

- `forEach()` accepts a function that performs an operation on each element:

```
map.forEach((k, v) -> System.out.println(v));
```

- `compute()` creates or updates the current value to the result of the calculation (it is possible to use the key and the current value):

```
map.compute("a", (k, v) -> String.valueOf(k).concat(v)); //["a", "aAa"]
```

- `computeIfPresent()` if the key exists, updates the current value to the result of the calculation (it is possible to use the key and the current value):

```
map.computeIfPresent("a", (k, v) -> k.concat(v));
```

- `computeIfAbsent()` if the key is missing, creates it with the value that is calculated (it is possible to use the key):

```
map.computeIfAbsent("a", k -> "A".concat(k)); //["a","Aa"]
```

- `getOrDefault()` if there is no key, returns the passed value by default:

```
map.getOrDefault("a", "not found");
```

- `merge()` accepts a key, a value, and a function that combines the transmitted and current values. If there is no value under the specified key, then it writes the transmitted value there.

```
map.merge("a", "z", (value, newValue) -> value.concat(newValue)); //["a","
```

↑ back to top

Q. What is LocalDateTime?

`LocalDateTime`combines together `LocaleDate`and `LocalTime`contains the date and time in the calendar system ISO-8601 without reference to the time zone. Time is stored accurate to the nanosecond. It contains many convenient methods such as plusMinutes, plusHours, isAfter, toSecondOfDay, etc.

↥ back to top

Q. What is ZonedDateTime?

`java.time.ZonedDateTime`- an analogue `java.util.Calendar`, a class with the most complete amount of information about the temporary context in the calendar system ISO-8601. It includes a time zone, therefore, this class carries out all operations with time shifts taking into account it.

↥ back to top

Q. How to get current date using Date Time API from Java 8?

```
LocalDate as.now();
```

↥ back to top

Q. How to add 1 week, 1 month, 1 year, 10 years to the current date using the Date Time API?

```
LocalDate as.now ().plusWeeks ( 1 );
LocalDate as.now ().plusMonths ( 1 );
LocalDate as.now ().plusYears ( 1 );
LocalDate as.now ().plus ( 1 , ChronoUnit.DECADES );
```

↥ back to top

Q. How to get the next Tuesday using the Date Time API?

```
LocalDate as.now().with( TemporalAdjusters.next ( DayOfWeek.TUESDAY ));
```

↥ back to top

Q. How to get the current time accurate to milliseconds using the Date Time API?

```
new  Date ().toInstant ();
```

↥ back to top

Q. How to get the second Saturday of the current month using the Date Time API?

```
LocalDate
    .of ( LocalDate.Now ().GetYear (), LocalDate.Now ().GetMonth (), 1 )
    .with ( TemporalAdjusters.nextOrSame ( DayOfWeek.SATURDAY ))
    .with ( TemporalAdjusters.next ( DayOfWeek.SATURDAY ));
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. How to get the current time in local time accurate to milliseconds using the Date Time API?

```
LocalDateTime.ofInstant ( new  Date().toInstant(), ZoneId.systemDefault())
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. How to determine repeatable annotation?

To define a repeatable annotation, you must create a container annotation for the list of repeatable annotations and designate a repeatable meta annotation `@Repeatable`:

```
@interface  Schedulers {
    Scheduler [] value ();
}

@Repeatable ( Schedulers . Class)
 @interface  Scheduler {
    String  birthday () default "Jan 8 2000";
 }
```

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is Nashorn?

Nashorn is a JavaScript engine developed in Java by Oracle. Designed to provide the ability to embed JavaScript code in Java applications. Compared to Rhino , which is supported by the Mozilla Foundation, Nashorn provides 2 to 10 times better performance, as it compiles code and transfers bytecode to the Java virtual machine directly in memory. Nashorn can compile JavaScript code and generate Java classes that are loaded with a special loader. It is also possible to call Java code directly from JavaScript.

```
<b><a href="#">↥ back to top</a></b>
```

Q. What is jjs?

`jjs` - This is a command line utility that allows you to execute JavaScript programs directly in the console.

```
<b><a href="#">↥ back to top</a></b>
```

Q. What class appeared in Java 8 for encoding / decoding data?

`Base64`- a thread-safe class that implements a data encoder and decoder using a base64 encoding scheme according to RFC 4648 and RFC 2045 .

Base64 contains 6 basic methods:

`getEncoder() / getDecoder()`- returns a base64 encoder / decoder conforming to the RFC 4648 standard ; getUrlEncoder()/ `getUrlDecoder()`- returns URL-safe base64 encoder / decoder conforming to RFC 4648 standard ; `getMimeEncoder() / getMimeDecoder()`- returns a MIME encoder / decoder conforming to RFC 2045 .

```
<b><a href="#">↥ back to top</a></b>
```

Q. How to create a Base64 encoder and decoder?

```
// Encode
String b64 =  Base64.getEncoder().encodeToString ( " input " . getBytes (
// Decode
new  String ( Base64.getDecoder().decode ( " aW5wdXQ == " ), " utf-8 " );
```

↥ back to top

Q. What are the functional interfaces `Function<T,R>`, `DoubleFunction<R>`, `IntFunction<R>` and `LongFunction<R>`?

`Function<T, R>`- the interface with which a function is implemented that receives an instance of the class `T` and returns an instance of the class at the output `R`.

Default methods can be used to build call chains (`compose`, `andThen`).

```
Function < String , Integer > toInteger =  Integer :: valueOf;
Function < String , String > backToString = toInteger.andThen ( String ::
backToString.apply("123");     // "123"
```

- `DoubleFunction<R>`- a function that receives input `Double` and returns an instance of the class at the output `R`;
- `IntFunction<R>`- a function that receives input `Integer`and returns an instance of the class at the output `R`;
- `LongFunction<R>`- a function that receives input `Long`and returns an instance of the class at the output `R`.

↥ back to top

Q. What are the functional interfaces `UnaryOperator<T>`, `DoubleUnaryOperator`, `IntUnaryOperator`and `LongUnaryOperator`?

`UnaryOperator<T>`(**unary operator**) takes an object of type as a parameter `T`, performs operations on them and returns the result of operations in the form of an object of type `T`:

```
UnaryOperator < Integer > operator = x - > x * x;
System.out.println(operator.apply ( 5 )); // 25
```

- `DoubleUnaryOperator`- unary operator receiving input `Double`;
- `IntUnaryOperator`- unary operator receiving input `Integer`;
- `LongUnaryOperator`- unary operator receiving input `Long`.

↥ back to top

Q. What are the functional interfaces `BinaryOperator<T>`, `DoubleBinaryOperator`, `IntBinaryOperator`and `LongBinaryOperator`?

`BinaryOperator<T>`(**binary operator**) - an interface through which a function is implemented that receives two instances of the class `T`and returns an instance of the class at the output `T`.

```
BinaryOperator < Integer > operator = (a, b) -> a + b;
System.out.println(operator.apply ( 1 , 2 )); // 3
```

- `DoubleBinaryOperator`- binary operator receiving input Double;
- `IntBinaryOperator`- binary operator receiving input Integer;
- `LongBinaryOperator`- binary operator receiving input Long.

↥ back to top

Q. What are the functional interfaces `Predicate<T>`, `DoublePredicate`, `IntPredicate`and `LongPredicate`?

`Predicate<T>`(**predicate**) - the interface with which a function is implemented that receives an instance of the class as input `T`and returns the type value at the output `boolean`.

The interface contains a variety of methods by default, allow to build complex conditions (`and`, `or`, `negate`).

```
Predicate < String > predicate = (s) -> s.length () >  0 ;
predicate.test("foo"); // true
predicate.negate().test("foo"); // false
```

- `DoublePredicate`- predicate receiving input `Double`;
- `IntPredicate`- predicate receiving input `Integer`;
- `LongPredicate`- predicate receiving input `Long`.

↥ back to top

Q. What are the functional interfaces `Consumer<T>`, `DoubleConsumer`, `IntConsumer`and `LongConsumer`?

`Consumer<T>`(**consumer**) - the interface through which a function is implemented that receives an instance of the class as an input `T`, performs some action with it, and returns nothing.

```
Consumer<String> hello = (name) ->  System.out.println( " Hello, "  + name
hello.accept( " world " );
```

- `DoubleConsumer`- the consumer receiving the input `Double`;
- `IntConsumer`- the consumer receiving the input `Integer`;
- `LongConsumer`- the consumer receiving the input `Long`.

↥ back to top

Q. What are the functional interfaces `Supplier<T>`, `BooleanSupplier`, `DoubleSupplier`, `IntSupplier`and `LongSupplier`?

`Supplier<T>`(**provider**) - the interface through which a function is implemented that takes nothing to the input, but returns the result of the class to the output `T`;

```
Supplier < LocalDateTime > now =  LocalDateTime::now;
now.get();
```

- `DoubleSupplier`- the supplier is returning `Double`;
- `IntSupplier`- the supplier is returning `Integer`;
- `LongSupplier`- the supplier is returning `Long`.

↥ back to top

Q. When do we go for Java 8 Stream API?

Q. Why do we need to use Java 8 Stream API in our projects?

Q. Explain Differences between Collection API and Stream API?

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

Q. What is Optional in Java 8? What is the use of Optional?

Q. 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?

Q. What is differences between Functional Programming and Object-Oriented Programming?

```
<b><a href="#">↥ back to top</a></b>
```

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project