

Java SE 8 New Features

Study Notes



December 1, 2015

Vladimir Virlan

Contents

[Introduction 2](#_Toc436833898)

[Java Programming Language 2](#_Toc436833899)

[Lambda Expressions 2](#_Toc436833900)

[Functional Interfaces 2](#_Toc436833901)

[Predicate<Person> predicate 3](#_Toc436833902)

[Consumer<Person> block 3](#_Toc436833903)

[Function<T,R> 3](#_Toc436833904)

[Supplier<T> 3](#_Toc436833905)

[Method references 3](#_Toc436833906)

[Reference to a Static Method 3](#_Toc436833907)

[Reference to an Instance Method of a Particular Object 3](#_Toc436833908)

[Reference to an Instance Method of an Arbitrary Object of a Particular Type 3](#_Toc436833909)

[Reference to a Constructor 4](#_Toc436833910)

[Default methods and statics in interfaces 4](#_Toc436833911)

[Repeating Annotations 4](#_Toc436833912)

[Type Annotations 4](#_Toc436833913)

[Improved type inference 4](#_Toc436833914)

[Method parameter reflection 4](#_Toc436833915)

[java.lang and java.util Packages 4](#_Toc436833916)

[Standard Encoding and Decoding Base64 4](#_Toc436833917)

[Unsigned Arithmetic Support 9](#_Toc436833918)

[Stream API 9](#_Toc436833919)

[Aggregate Operations 9](#_Toc436833920)

[Reduction 10](#_Toc436833921)

[Parallelism 10](#_Toc436833922)

[Date Time API 10](#_Toc436833923)

[Optional 10](#_Toc436833924)

[Concurrency 11](#_Toc436833925)

[Tools 11](#_Toc436833926)

[jdeps 11](#_Toc436833927)

[Nashorn, JavaScript Engine 11](#_Toc436833928)

# Introduction

This document contains my study notes for the new features in Java SE 8.

# Java Programming Language

## Lambda Expressions

<https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>

### Functional Interfaces

*Functional interfaces* provide target types for lambda expressions and method references. Each functional interface has a single abstract method, called the *functional method* for that functional interface, to which the lambda expression's parameter and return types are matched or adapted. Functional interfaces can provide a target type in multiple contexts, such as assignment context, method invocation, or cast context:

// Assignment context

Predicate<String> p = String::isEmpty;

// Method invocation context

stream.filter(e -> e.getSize() > 10)...

// Cast context

stream.map((ToIntFunction) e -> e.getSize())...

The interfaces in this package are general purpose functional interfaces used by the JDK, and are available to be used by user code as well. While they do not identify a complete set of function shapes to which lambda expressions might be adapted, they provide enough to cover common requirements. Other functional interfaces provided for specific purposes, such as [FileFilter](https://docs.oracle.com/javase/8/docs/api/java/io/FileFilter.html), are defined in the packages where they are used.

The interfaces in this package are annotated with [FunctionalInterface](https://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html). This annotation is not a requirement for the compiler to recognize an interface as a functional interface, but merely an aid to capture design intent and enlist the help of the compiler in identifying accidental violations of design intent.

Functional interfaces often represent abstract concepts like functions, actions, or predicates. In documenting functional interfaces, or referring to variables typed as functional interfaces, it is common to refer directly to those abstract concepts, for example using "this function" instead of "the function represented by this object". When an API method is said to accept or return a functional interface in this manner, such as "applies the provided function to...", this is understood to mean a *non-null* reference to an object implementing the appropriate functional interface, unless potential nullity is explicitly specified.

### Predicate<Person> predicate

### Consumer<Person> block

### Function<T,R>

### Supplier<T>

## Method references

<https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>

There are four kinds of method references:

|  |  |
| --- | --- |
| **Kind** | **Example** |
| Reference to a static method | ContainingClass::staticMethodName |
| Reference to an instance method of a particular object | containingObject::instanceMethodName |
| Reference to an instance method of an arbitrary object of a particular type | ContainingType::methodName |
| Reference to a constructor | ClassName::new |

### Reference to a Static Method

Here you can reference a static method from your class like this: YourClass::theStaticMethod. In our case it is: Car::compareByMileage. But this makes sense in a context because you are not providing the method arguments: Arrays.sort(carsArray, Car::compareByMileage); The method reference Car::compareByMileage is equivalent to the lambda expression:(a,b) -> Car.compareByMileage(a,b)

### Reference to an Instance Method of a Particular Object

Here is an example of accessing a non-static method on an instance. Again the context is important. You can do this only where the method is expecting a functional interface.

ComparisonProvider comparisonProvider = **new** ComparisonProvider();

Arrays.*sort*(carsArray, comparisonProvider::compareByBrand);

Here, comparisonProvider.compareByBrand would be:

**public** **class** ComparisonProvider {

**public** **int** compareByBrand(Car a, Car b) {

**return** a.getBrand().compareTo(b.getBrand());

}

}

In this case, comparisonProvider::compareByBrand invokes the compareByName method and JRE infers the method arguments (Car, Car).

### Reference to an Instance Method of an Arbitrary Object of a Particular Type

Here we show an example of an instance method of an arbitrary type:

String[] array = { "Zoi", "Wow", "Aoc", "Dao", "Fao", "Aao" };

Arrays.*sort*(array, String::compareToIgnoreCase);

### Reference to a Constructor

You can refer to constructors just like you would refer to a static method:

Set<Car> carsSet = transferElements(cars, HashSet::new);

The Java compiler infers that you want to create a HashSet collection that contains elements of type Person. Alternatively, you can specify this as follows:

Set<Car> carsSetNew = *transferElements*(cars, HashSet<Car>::**new**);

This is equivalent to:

Set<Car> carsSetLambda = *transferElements*(cars, ()->{**return** **new** HashSet<>();});

Where transferElements is:

**public** **static** <T, SOURCE **extends** Collection<T>, DEST **extends** Collection<T>>

DEST transferElements(SOURCE sourceCollection, Supplier<DEST> collectionFactory) {

DEST result = collectionFactory.get();

**for** (T t : sourceCollection) {

result.add(t);

}

**return** result;

}

The functional interface Supplier contains one method get that takes no arguments and returns an object.

## Default methods and statics in interfaces

<https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html>

## Repeating Annotations

## Type Annotations

## Improved type inference

## Method parameter reflection

# java.lang and java.util Packages

## Standard Encoding and Decoding Base64

<https://en.wikipedia.org/wiki/Base64>

A quote from Thomas Hobbes' *Leviathan* (be aware of spaces between lines):

Man is distinguished, not only by his reason, but by this singular passion from

other animals, which is a lust of the mind, that by a perseverance of delight

in the continued and indefatigable generation of knowledge, exceeds the short

vehemence of any carnal pleasure.

is represented as a byte sequence of 8-bit-padded [ASCII](https://en.wikipedia.org/wiki/ASCII) characters encoded in [MIME](https://en.wikipedia.org/wiki/MIME)'s Base64 scheme as follows:

TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz

IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg

dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu

dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo

ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=

In the above quote, the encoded value of *Man* is *TWFu*. Encoded in ASCII, the characters *M*, *a*, and *n* are stored as the bytes 77, 97, and 110, which are the 8-bit binary values 01001101, 01100001, and 01101110. These three values are joined together into a 24-bit string, producing010011010110000101101110. Groups of 6 bits (6 bits have a maximum of 26 = 64 different binary values) are [converted into individual numbers](https://en.wikipedia.org/wiki/Binary_number#Counting_in_binary) from left to right (in this case, there are four numbers in a 24-bit string), which are then converted into their corresponding Base64 character values.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Text content** | **M** | | | | | | | | **a** | | | | | | | | **n** | | | | | | | |
| **ASCII** | 77 (0x4d) | | | | | | | | 97 (0x61) | | | | | | | | 110 (0x6e) | | | | | | | |
| **Bit pattern** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| **Index** | 19 | | | | | | 22 | | | | | | 5 | | | | | | 46 | | | | | |
| **Base64-encoded** | **T** | | | | | | **W** | | | | | | **F** | | | | | | **u** | | | | | |

As this example illustrates, Base64 encoding converts three [octets](https://en.wikipedia.org/wiki/Octet_(computing)) into four encoded characters.

The Base64 index table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value** | **Char** |  | **Value** | **Char** |  | **Value** | **Char** |  | **Value** | **Char** |
| 0 | A | 16 | Q | 32 | g | 48 | w |
| 1 | B | 17 | R | 33 | h | 49 | x |
| 2 | C | 18 | S | 34 | i | 50 | y |
| 3 | D | 19 | T | 35 | j | 51 | z |
| 4 | E | 20 | U | 36 | k | 52 | 0 |
| 5 | F | 21 | V | 37 | l | 53 | 1 |
| 6 | G | 22 | W | 38 | m | 54 | 2 |
| 7 | H | 23 | X | 39 | n | 55 | 3 |
| 8 | I | 24 | Y | 40 | o | 56 | 4 |
| 9 | J | 25 | Z | 41 | p | 57 | 5 |
| 10 | K | 26 | a | 42 | q | 58 | 6 |
| 11 | L | 27 | b | 43 | r | 59 | 7 |
| 12 | M | 28 | c | 44 | s | 60 | 8 |
| 13 | N | 29 | d | 45 | t | 61 | 9 |
| 14 | O | 30 | e | 46 | u | 62 | + |
| 15 | P | 31 | f | 47 | v | 63 | / |

When the number of bytes to encode is not divisible by three (that is, if there are only one or two bytes of input for the last 24-bit block), then the following action is performed:

Add extra bytes with value zero so there are three bytes, and perform the conversion to base64. If there was only one significant input byte, only the first two base64 digits are picked (12 bits), and if there were two significant input bytes, the first three base64 digits are picked (18 bits). '=' characters might be added to make the last block contain four base64 characters.

As a result, when the last group contains one octet, the four [least significant bits](https://en.wikipedia.org/wiki/Least_significant_bit) of the final 6-bit block are set to zero; and when the last group contains two octets, the two least significant bits of the final 6-bit block are set to zero.

Here is an example of how to use Encoder and Decoder in Java 8:

**package** com.vvirlan;

**import** java.net.URL;

**import** java.**util**.Base64;

**class** Base64Sample {

**public** **static** **void** main(String args[]) {

**try** {

// Encode URL

URL originalUrl = **new** URL("http://example.com");

Base64.Encoder urlEncoder = Base64.*getUrlEncoder*();

System.***out***.println("URL: " + originalUrl);

**byte**[] encodedUrl = urlEncoder.encode(originalUrl.toString().getBytes("UTF8"));

System.***out***.println("Base64 Encoded URL : " + **new** String(encodedUrl, "UTF-8"));

// Encode text

String originalText = "thisIsAnExampleString";

Base64.Encoder textEncoder = Base64.*getEncoder*();

**byte**[] encodedText = textEncoder.encode(originalText.getBytes("UTF-8"));

System.***out***.println("Base64 Encoded String : " + **new** String(encodedText, "UTF-8"));

// Decode URL and text

Base64.Decoder urlDecoder = Base64.*getUrlDecoder*();

Base64.Decoder textDecoder = Base64.*getDecoder*();

**byte**[] urlDecoded = urlDecoder.decode(encodedUrl);

**byte**[] textDecoded = textDecoder.decode(encodedText);

System.***out***.println("Base64 Decoded URL : " + **new** String(urlDecoded, "UTF-8"));

System.***out***.println("Base64 Decoded String : " + **new** String(textDecoded, "UTF-8"));

}

**catch** (Exception e) {

System.***out***.println("Invalid URL Exception");

}

}

}

Here are the APIs for **Base64**:

public class **Base64** extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)

This class consists exclusively of static methods for obtaining encoders and decoders for the Base64 encoding scheme. The implementation of this class supports the following types of Base64 as specified in [RFC 4648](http://www.ietf.org/rfc/rfc4648.txt) and [RFC 2045](http://www.ietf.org/rfc/rfc2045.txt).

**Basic**

Uses "The Base64 Alphabet" as specified in Table 1 of RFC 4648 and RFC 2045 for encoding and decoding operation. The encoder does not add any line feed (line separator) character. The decoder rejects data that contains characters outside the base64 alphabet.

**URL and Filename safe**

Uses the "URL and Filename safe Base64 Alphabet" as specified in Table 2 of RFC 4648 for encoding and decoding. The encoder does not add any line feed (line separator) character. The decoder rejects data that contains characters outside the base64 alphabet.

**MIME**

Uses the "The Base64 Alphabet" as specified in Table 1 of RFC 2045 for encoding and decoding operation. The encoded output must be represented in lines of no more than 76 characters each and uses a carriage return '\r' followed immediately by a linefeed '\n' as the line separator. No line separator is added to the end of the encoded output. All line separators or other characters not found in the base64 alphabet table are ignored in decoding operation.

Unless otherwise noted, passing a null argument to a method of this class will cause a [NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) to be thrown.

|  |  |
| --- | --- |
| static [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) | [**getDecoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getDecoder--)()  Returns a [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) that decodes using the [**Basic**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#basic) type base64 encoding scheme. |
| static [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) | [**getEncoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getEncoder--)()  Returns a [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) that encodes using the [**Basic**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#basic) type base64 encoding scheme. |
| static [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) | [**getMimeDecoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getMimeDecoder--)()  Returns a [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) that decodes using the [**MIME**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#mime) type base64 decoding scheme. |
| static [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) | [**getMimeEncoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getMimeEncoder--)()  Returns a [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) that encodes using the [**MIME**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#mime) type base64 encoding scheme. |
| static [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) | [**getMimeEncoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getMimeEncoder-int-byte:A-)(int lineLength, byte[] lineSeparator)  Returns a [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) that encodes using the [**MIME**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#mime) type base64 encoding scheme with specified line length and line separators. |
| static [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) | [**getUrlDecoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getUrlDecoder--)()  Returns a [**Base64.Decoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) that decodes using the [**URL and Filename safe**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#url) type base64 encoding scheme. |
| static [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) | [**getUrlEncoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#getUrlEncoder--)()  Returns a [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) that encodes using the [**URL and Filename safe**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html#url) type base64 encoding scheme. |

public static class **Base64.Decoder** extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)

This class implements a decoder for decoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045. The Base64 padding character '=' is accepted and interpreted as the end of the encoded byte data, but is not required. So if the final unit of the encoded byte data only has two or three Base64 characters (without the corresponding padding character(s) padded), they are decoded as if followed by padding character(s). If there is a padding character present in the final unit, the correct number of padding character(s) must be present, otherwise IllegalArgumentException (IOException when reading from a Base64 stream) is thrown during decoding.

Instances of [Base64.Decoder](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html) class are safe for use by multiple concurrent threads.

Unless otherwise noted, passing a null argument to a method of this class will cause a [NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) to be thrown.

|  |  |
| --- | --- |
| byte[] | [**decode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html#decode-byte:A-)(byte[] src)  Decodes all bytes from the input byte array using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme, writing the results into a newly-allocated output byte array. |
| int | [**decode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html#decode-byte:A-byte:A-)(byte[] src, byte[] dst)  Decodes all bytes from the input byte array using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme, writing the results into the given output byte array, starting at offset 0. |
| [**ByteBuffer**](https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html) | [**decode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html#decode-java.nio.ByteBuffer-)([**ByteBuffer**](https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html) buffer)  Decodes all bytes from the input byte buffer using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme, writing the results into a newly-allocated ByteBuffer. |
| byte[] | [**decode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html#decode-java.lang.String-)([**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) src)  Decodes a Base64 encoded String into a newly-allocated byte array using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme. |
| [**InputStream**](https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html) | [**wrap**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Decoder.html#wrap-java.io.InputStream-)([**InputStream**](https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html) is)  Returns an input stream for decoding [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoded byte stream. |

public static class **Base64.Encoder** extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)

This class implements an encoder for encoding byte data using the Base64 encoding scheme as specified in RFC 4648 and RFC 2045. Instances of [Base64.Encoder](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) class are safe for use by multiple concurrent threads. Unless otherwise noted, passing a null argument to a method of this class will cause a [NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) to be thrown.

|  |  |
| --- | --- |
| byte[] | [**encode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#encode-byte:A-)(byte[] src)  Encodes all bytes from the specified byte array into a newly-allocated byte array using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme. |
| int | [**encode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#encode-byte:A-byte:A-)(byte[] src, byte[] dst)  Encodes all bytes from the specified byte array using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme, writing the resulting bytes to the given output byte array, starting at offset 0. |
| [**ByteBuffer**](https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html) | [**encode**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#encode-java.nio.ByteBuffer-)([**ByteBuffer**](https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html) buffer)  Encodes all remaining bytes from the specified byte buffer into a newly-allocated ByteBuffer using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html)encoding scheme. |
| [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) | [**encodeToString**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#encodeToString-byte:A-)(byte[] src)  Encodes the specified byte array into a String using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme. |
| [**Base64.Encoder**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html) | [**withoutPadding**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#withoutPadding--)()  Returns an encoder instance that encodes equivalently to this one, but without adding any padding character at the end of the encoded byte data. |
| [**OutputStream**](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html) | [**wrap**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.Encoder.html#wrap-java.io.OutputStream-)([**OutputStream**](https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html) os)  Wraps an output stream for encoding byte data using the [**Base64**](https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html) encoding scheme. |

## Unsigned Arithmetic Support

# Stream API

## Aggregate Operations

<https://docs.oracle.com/javase/tutorial/collections/streams/index.html>

**Differences Between Aggregate Operations and Iterators**

Aggregate operations, like forEach, appear to be like iterators. However, they have several fundamental differences:

* **They use internal iteration**: Aggregate operations do not contain a method like next to instruct them to process the next element of the collection. With internal delegation, your application determines what collection it iterates, but the JDK determines how to iterate the collection. With external iteration, your application determines both what collection it iterates and how it iterates it. However, external iteration can only iterate over the elements of a collection sequentially. Internal iteration does not have this limitation. It can more easily take advantage of parallel computing, which involves dividing a problem into sub-problems, solving those problems simultaneously, and then combining the results of the solutions to the sub-problems. See the section [Parallelism](https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html) for more information.
* **They process elements from a stream**: Aggregate operations process elements from a stream, not directly from a collection. Consequently, they are also called stream operations.
* **They support behavior as parameters**: You can specify [lambda expressions](https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html) as parameters for most aggregate operations. This enables you to customize the behavior of a particular aggregate operation.

## Reduction

<https://docs.oracle.com/javase/tutorial/collections/streams/reduction.html>

Map<Person.Sex, List<Person>> byGender =

persons

.stream()

.collect(Collectors.*groupingBy*(Person::getSex));

System.***out***.println(byGender);

## Parallelism

<https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>

An example of parallel stream processing:

Date start = **new** Date();

**double** average = persons

.parallelStream()

.filter(p->p.getSex().equals(Person.Sex.***MALE***))

.mapToInt(Person::getAge)

.average()

.getAsDouble();

Date end = **new** Date();

**long** diff = end.getTime() - start.getTime();

System.***out***.println("Average age of Males: "+average+" Time: "+diff+" ms.");

# Date Time API

# Optional

# Concurrency

# Tools

## jdeps

# Nashorn, JavaScript Engine