JAVA 8

First Bite

October 20, 2015



# SAM interfaces

**@FunctionalInterface** // (can only contain Single abstract method) in Java 7 called SAM interfaces

**public** **interface** TestInterface {

**public** **void** test();

}

**public** **class** TestJava8 {

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

// An instance of a functional interface using a lambda expression

TestInterface tester = () -> System.***out***.println("Java SE 8 is working!");

tester.test();

}

}

Runnable r1 = **new** Runnable() {

@Override

**public** **void** run() {

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("Running Thread 1");

}

};

Runnable r2 = **new** Runnable() {

@Override

**public** **void** run() {

System.***out***.println("Running Thread 2");

}

};

**new** Thread(r1).start();

**new** Thread(r2).start();

Runnable r1 = () -> {

**try** {

Thread.sleep(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.**out**.println("Running Thread 1");

};

Runnable r2 = () -> System.***out***.println("Running Thread 2");

**new** Thread(r1).start();

**new** Thread(r2).start();

List<String> strings = **new** ArrayList<String>();

strings.add("bbb");

strings.add("ddd");

strings.add("EEE");

strings.add("CCC");

strings.add("AAA");

//Case-insensitive sort with an anonymous class

Collections.*sort*(strings, **new** Comparator<String>() {

@Override

**public** **int** compare(String str1, String str2) {

**return** str1.compareToIgnoreCase(str2);

}

});

System.***out***.println("Sort with comparator");

**for**(String str: strings){

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

}

//Case-insensitive sort with an anonymous class

Comparator<String> comp = (str1, str2) -> {

**return** str1.compareToIgnoreCase(str2);

};

Collections.*sort*(strings, comp);

System.***out***.println("Sort with comparator");

**for**(String str: strings){

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

}

**for**(String str: strings){

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

}

strings.forEach(str -> System.***out***.println(str));

//Traverse with iterator

Iterator<String> i = strings.iterator();

**while** (i.hasNext()) {

System.***out***.println(i.next());

}

strings.forEach(str -> System.***out***.println(str));

# Predicate Interface

Represents a predicate (boolean-valued function) of one argument. This is a functional interface whose functional method is test(Object).

**Type Parameters:**

**<T>** the type of the input to the predicate

**Since:**

1.8

List<Person> people = **new** ArrayList<>();

people.add(**new** Person("Joe", 48));

people.add(**new** Person("Mary", 30));

people.add(**new** Person("Mike", 73));

Predicate<Person> pred = **new** Predicate<Person>() {

@Override

**public** **boolean** test(Person p) {

**return** (p.getAge() >= 65);

}

};

**for** (Person p : people) {

**if**(pred.test(p)){

System.***out***.println(p.getPersonInfo());

}

}

Predicate<Person> pred = (p) -> p.getAge() > 65;

people.forEach(p -> {

**if** (pred.test(p))

{

System.***out***.println(p.getPersonInfo());

}

});

# Method References

* Static Method References
* Instance Method References

## Static Method References

**public** **class** Person {

**private** String name;

**private** **int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** name + " (" + age + ")";

}

**public** **static** **int** **compareAges**(Person p1, Person p2) {

Integer age1 = p1.getAge();

**return** age1.compareTo(p2.getAge());

}

}

List<Person> people = **new** ArrayList<>();

people.add(**new** Person("Joe", 48));

people.add(**new** Person("Mary", 30));

people.add(**new** Person("Mike", 73));

Collections.*sort*(people, **Person :: *compareAges***);

people.forEach(p -> System.***out***.println(p) );

## Instance Method References – In the same class

**public** **int** **compareAges**(Person p1, Person p2) {

Integer age1 = p1.getAge();

**return** age1.compareTo(p2.getAge());

}

Collections.*sort*(people, **this** :: **compareAges**);

people.forEach(p -> System.***out***.println(p) );

# Default Methods

**default** String getPersonInfo() {

**return** getName() + " (" + getAge() + ")";

}

Old Java Versions

Interface -> Base class -> sub classes

With Java 8, using default methods

Interface -> ~~Base class~~ -> sub classes

# Implementing static methods in interfaces

**public** **interface** PersonInterface {

String getName();

**void** setName(String name);

**int** getAge();

**void** setAge(**int** age);

**static** String getPersonInfo(Person p) {

**return** p.getName() + " (" + p.getAge() + ")";

}

}

**public** **class** UseStaticMethod {

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

List<Person> people = **new** ArrayList<>();

people.add(**new** Person("Joe", 48));

people.add(**new** Person("Mary", 30));

people.add(**new** Person("Mike", 73));

Predicate<Person> pred = (p) -> p.getAge() > 65;

*displayPeople*(people, pred);

}

**private** **static** **void** displayPeople(List<Person> people,

Predicate<Person> pred) {

System.***out***.println("Selected:");

people.forEach(p -> {

**if**(pred.test(p))

{

String info = **PersonInterface.*getPersonInfo*(p);**

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

}

});

# }

# Streams

* Sequential Stream
* Parallel Stream

## Sequential Stream

people.forEach(p -> {

**if** (pred.test(p))

{

System.***out***.println(p.getName());

}

});

people.stream()

.forEach(p -> System.***out***.println(p.getName()));

people.stream()

**.filter(pred)**

.forEach(p -> System.***out***.println(p.getName()));

## Parallel Stream – For large data set / Multi core machines

people.**parallelStream()**

.filter(pred)

.forEach(p -> System.***out***.println(p.getName()));

people.stream()

**.parallel()**

.filter(pred)

.forEach(p -> System.***out***.println(p.getName()));

# Streams from collections and arrays

Person[] people = {

**new** Person("Joe", 48),

**new** Person("Mary", 30),

**new** Person("Mike", 73)

};

Stream<Person> stream = Stream.*of*(people);

Stream<Person> stream = Arrays.*stream*(people);

stream.forEach(p -> System.***out***.println(p.getInfo()));

# Aggregating stream values

**long** count = **strings.stream().parallel().count();**

List<Person> people = **new** ArrayList<>();

people.add(**new** Person("Joe", 48));

people.add(**new** Person("Mary", 30));

people.add(**new** Person("Mike", 73));

**int** sum = people.stream()

.mapToInt(p -> p.getAge())

.sum();

System.***out***.println("Total of ages: " + sum);

OptionalDouble avg = people.stream()

.mapToInt(p -> p.getAge())

.average();

**if** (avg.isPresent()) {

System.***out***.println("Average: " + avg.getAsDouble());

} **else** {

System.***out***.println("Average not calculated");

}

**int** sum = people.parallelStream()

.mapToInt(p -> p.getAge())

.sum();

System.***out***.println("Total of ages: " + sum);

OptionalDouble avg = people.parallelStream()

.mapToInt(p -> p.getAge())

.average();

**if** (avg.isPresent()) {

System.***out***.println("Average: " + avg.getAsDouble());

} **else** {

System.***out***.println("Average not calculated");

}

# New Dates And Times API

## Instant And Duration

**import** java.time.Duration;

**import** java.time.Instant;

Instant start = Instant.*now*();

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

Thread.*sleep*(1000);

Instant end = Instant.*now*();

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

Duration elapsed = Duration.*between*(start, end);

System.***out***.println("Elapsed: " + elapsed.toMillis() + " milliseconds");

System.***out***.println("Elapsed: " + elapsed.toNanos() + " nanoseconds");

## Dates And Times

**import** java.time.LocalDate;

**import** java.time.LocalDateTime;

**import** java.time.LocalTime;

LocalDate currentDate = LocalDate.*now*();

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

LocalDate specificDate = LocalDate.*of*(2000, 1, 1);

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

LocalTime currentTime = LocalTime.*now*();

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

LocalTime specificTime = LocalTime.*of*(14, 0, 45);

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

LocalDateTime currentDT = LocalDateTime.*now*();

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

LocalDateTime specificDT = LocalDateTime.*of*(specificDate, specificTime);

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

## Format Date Time

LocalDate currentDate = LocalDate.*now*();

DateTimeFormatter df = DateTimeFormatter.***ISO\_DATE***;

System.***out***.println("ISO\_DATE: "+df.format(currentDate));

LocalTime currentTime = LocalTime.*now*();

DateTimeFormatter tf = DateTimeFormatter.***ISO\_TIME***;

System.***out***.println("ISO\_TIME: "+tf.format(currentTime));

LocalDateTime currentDT = LocalDateTime.*now*();

DateTimeFormatter dtf = DateTimeFormatter.***ISO\_DATE\_TIME***;

System.***out***.println("ISO\_DATE\_TIME: "+dtf.format(currentDT));

DateTimeFormatter f\_long = DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***LONG***);

System.***out***.println("FormatStyle.LONG: "+f\_long.format(currentDT));

DateTimeFormatter f\_short = DateTimeFormatter.*ofLocalizedDate*(FormatStyle.***SHORT***);

System.***out***.println("FormatStyle.SHORT: "+f\_short.format(currentDT));

String fr\_short = f\_short.withLocale(Locale.***FRENCH***).format(currentDT);

String fr\_long = f\_long.withLocale(Locale.***FRENCH***).format(currentDT);

System.***out***.println("Locale.FRENCH SHORT: "+fr\_short);

System.***out***.println("Locale.FRENCH LONG: "+fr\_long);

String de\_short = f\_short.withLocale(Locale.***GERMAN***).format(currentDT);

String de\_long = f\_long.withLocale(Locale.***GERMAN***).format(currentDT);

System.***out***.println("Locale.GERMAN SHORT: "+de\_short);

System.***out***.println("Locale.GERMAN LONG: "+de\_long);

DateTimeFormatterBuilder b = **new** DateTimeFormatterBuilder()

.appendValue(ChronoField.***MONTH\_OF\_YEAR***)

.appendLiteral("||")

.appendValue(ChronoField.***DAY\_OF\_MONTH***)

.appendLiteral("||")

.appendValue(ChronoField.***YEAR***);

DateTimeFormatter f = b.toFormatter();

System.***out***.println("DateTimeFormatterBuilder - appendLiteral: "+f.format(currentDT));

## Time Zones

DateTimeFormatter dtf = DateTimeFormatter.*ofLocalizedDateTime*(FormatStyle.***SHORT***);

LocalDateTime dt = LocalDateTime.*now*();

System.***out***.println(dtf.format(dt));

ZonedDateTime gmt = ZonedDateTime.*now*(ZoneId.*of*("GMT+1"));

System.***out***.println(dtf.format(gmt));

ZonedDateTime gmt1 = ZonedDateTime.*now*(ZoneId.*of*("GMT"));

System.***out***.println(dtf.format(gmt1));

ZonedDateTime ny = ZonedDateTime.*now*(ZoneId.*of*("America/New\_York"));

System.***out***.println(dtf.format(ny));

ZonedDateTime ny1 = ZonedDateTime.*now*(ZoneId.*of*("Asia/Singapore"));

System.***out***.println(dtf.format(ny1));

Set<String> zones = ZoneId.*getAvailableZoneIds*();

//zones.forEach(z -> System.out.println(z));

Predicate<String> condition = str -> str.contains("London"); //Add a condition

zones.forEach(z -> {

**if** (condition.test(z)) {

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

}

});

# Java with JavaScript - Nashorn

## Running a JavaScript code within Java

ScriptEngineManager manager = **new** ScriptEngineManager();

ScriptEngine engine = manager.getEngineByName("nashorn");

String script = "var welcome = 'Hello'; "

+ "welcome += ', David'; "

+ "welcome;";

String result;

**try** {

result = (String)engine.eval(script);

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

} **catch** (ScriptException e) {

System.***out***.println("There was a JavaScript error");

e.printStackTrace();

}

## Run a JavaScript file within Java



**var** feed = 'http://services.explorecalifornia.org/rss/tours.php';

**var** url = **new** java.net.URL(feed);

input = **new** java.util.Scanner(url.openStream());

input.useDelimiter('$')

**var** contents = input.next()

contents

ScriptEngineManager mgr = **new** ScriptEngineManager();

ScriptEngine engine = mgr.getEngineByName("nashorn");

File f = **new** File("scripts/readurl.js");

Reader reader = **new** FileReader(f);

String result;

**try** {

result = (String)engine.eval(reader);

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

} **catch** (ScriptException e) {

System.***out***.println("There was a JavaScript error");

e.printStackTrace();

}

# Joining string values into delimited lists

## String.join

String stooges = String.*join*(" and ", "Larry", "Curly", "Moe");

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

String[] states = {"California", "Oregon", "Washington"};

String statesList = String.*join*(",", states);

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

LinkedList<String> strings1 = **new** LinkedList<String>();

strings1.add("Java");

strings1.add("is");

strings1.add("is");

strings1.add("cool");

String message1 = String.*join*(" ", strings1);

System.***out***.println(message1); // message returned is: "Java is is cool"

Set<String> strings2 = **new** LinkedHashSet<>();

strings2.add("Java");

strings2.add("is");

strings2.add("is");

strings2.add("very");

strings2.add("cool");

String message2 = String.*join*("-", strings2);

System.***out***.println(message2); // message returned is: "Java-is-very-cool"

## StringJoiner

//Creating a StringJoiner

StringJoiner sj = **new** StringJoiner(",", "{", "}");

sj.setEmptyValue("No stooges yet");

System.***out***.println(sj);//No stooges yet

sj.add("Moe")

.add("Larry")

.add("Curly");

System.***out***.println(sj);//{Moe,Larry,Curly}

//Merging StringJoiner instances

StringJoiner sj2 = **new** StringJoiner(",");

sj2.add("Shemp");

sj.merge(sj2);

System.***out***.println(sj);//{Moe,Larry,Curly,Shemp}

//Working with collections

Set<String> set = **new** TreeSet<>();

set.add("California");

set.add("Oregon");

set.add("Washington");

StringJoiner sj3 = **new** StringJoiner(" and ");

set.forEach( s -> sj3.add(s));

System.***out***.println(sj3);//California and Oregon and Washington

# Other Improvements

## Enhancements in Concurrency

* New methods to handle race conditions
* Improvements to ConcurrentHashMap
* Parallel operations with arrays
* New methods for composing async operations

## New Collection Methods

* Iterable : forEach()
* Collection : remdoveIf()
* List : replaceAll() , sort()
* Iterator : forEachRemaining()
* Map : forEach() , replace() , remove() …

## Encoding and Decoding Base64

* Base64 class has Encoder and Decoder member types
* Get an encoder or decoder instance from static methods
* Base 64 : getEncoder() , getMimeEncoder() , getUrlEncoder()
* Base 64 : getDecoder() , getMimeDecoder() , getUrlDecoder()
* Use encode() and decode() on String and byte array

## Improvement in Annotations

* Reduced redundancy
* Ability to annotate types
* Depends on definition of annotations in frameworks

## JDBC 4.2

* Date, Time and Timestamp can convert to java.time equivalents
* Statement class adds executeLargeUpdate() method – very large database update

## Minor Changes

* Object class has methods isNull and nonNull
* Lazy messages: reduce processing in code that might not be executed
* Regular expression improvements in Matscher and Pattern classes

More Details:

<http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>

# What's New in JDK 8

Java Platform, Standard Edition 8 is a major feature release. This document summarizes features and enhancements in Java SE 8 and in JDK 8, Oracle's implementation of Java SE 8. Click the component name for a more detailed description of the enhancements for that component.

* [Java Programming Language](http://docs.oracle.com/javase/8/docs/technotes/guides/language/enhancements.html#javase8)
  + Lambda Expressions, a new language feature, has been introduced in this release. They enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as functional interfaces) more compactly.
  + Method references provide easy-to-read lambda expressions for methods that already have a name.
  + Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
  + Repeating Annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
  + Type Annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
  + Improved type inference.
  + Method parameter reflection.
* [Collections](http://docs.oracle.com/javase/8/docs/technotes/guides/collections/changes8.html)
  + Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.
  + Performance Improvement for HashMaps with Key Collisions
* [Compact Profiles](http://docs.oracle.com/javase/8/docs/technotes/guides/compactprofiles/) contain predefined subsets of the Java SE platform and enable applications that do not require the entire Platform to be deployed and run on small devices.
* [Security](http://docs.oracle.com/javase/8/docs/technotes/guides/security/enhancements-8.html)
  + Client-side TLS 1.2 enabled by default
  + New variant of AccessController.doPrivileged that enables code to assert a subset of its privileges, without preventing the full traversal of the stack to check for other permissions
  + Stronger algorithms for password-based encryption
  + SSL/TLS Server Name Indication (SNI) Extension support in JSSE Server
  + Support for AEAD algorithms: The SunJCE provider is enhanced to support AES/GCM/NoPadding cipher implementation as well as GCM algorithm parameters. And the SunJSSE provider is enhanced to support AEAD mode based cipher suites. See Oracle Providers Documentation, JEP 115.
  + KeyStore enhancements, including the new Domain KeyStore type java.security.DomainLoadStoreParameter, and the new command option -importpassword for the keytool utility
  + SHA-224 Message Digests
  + Enhanced Support for NSA Suite B Cryptography
  + Better Support for High Entropy Random Number Generation
  + New java.security.cert.PKIXRevocationChecker class for configuring revocation checking of X.509 certificates
  + 64-bit PKCS11 for Windows
  + New rcache Types in Kerberos 5 Replay Caching
  + Support for Kerberos 5 Protocol Transition and Constrained Delegation
  + Kerberos 5 weak encryption types disabled by default
  + Unbound SASL for the GSS-API/Kerberos 5 mechanism
  + SASL service for multiple host names
  + JNI bridge to native JGSS on Mac OS X
  + Support for stronger strength ephemeral DH keys in the SunJSSE provider
  + Support for server-side cipher suites preference customization in JSSE
* [JavaFX](http://docs.oracle.com/javase/8/javase-clienttechnologies.htm)
  + The new Modena theme has been implemented in this release. For more information, see the blog at [fxexperience.com](http://fxexperience.com/2013/03/modena-theme-update/).
  + The new SwingNode class enables developers to embed Swing content into JavaFX applications. See the [SwingNode](http://docs.oracle.com/javase/8/javafx/api/javafx/embed/swing/SwingNode.html) javadoc and [Embedding Swing Content in JavaFX Applications](http://docs.oracle.com/javase/8/javafx/interoperability-tutorial/embed-swing.htm).
  + The new UI Controls include the [DatePicker](http://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/DatePicker.html) and the [TreeTableView](http://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TreeTableView.html) controls.
  + The javafx.print package provides the public classes for the JavaFX Printing API. See the [javadoc](http://docs.oracle.com/javase/8/javafx/api/javafx/print/package-summary.html) for more information.
  + The 3D Graphics features now include 3D shapes, camera, lights, subscene, material, picking, and antialiasing. The new Shape3D (Box, Cylinder, MeshView, and Sphere subclasses), SubScene, Material, PickResult, LightBase (AmbientLight and PointLight subclasses) , and SceneAntialiasing API classes have been added to the JavaFX 3D Graphics library. The Camera API class has also been updated in this release. See the corresponding class javadoc for javafx.scene.shape.Shape3D, javafx.scene.SubScene, javafx.scene.paint.Material, javafx.scene.input.PickResult, javafx.scene.SceneAntialiasing, and the [Getting Started with JavaFX 3D Graphics](http://docs.oracle.com/javase/8/javafx/graphics-tutorial/javafx-3d-graphics.htm) document.
  + The WebView class provides new features and improvements. Review [Supported Features of HTML5](http://docs.oracle.com/javase/8/javafx/embedded-browser-tutorial/index.html) for more information about additional HTML5 features including Web Sockets, Web Workers, and Web Fonts.
  + Enhanced text support including bi-directional text and complex text scripts such as Thai and Hindi in controls, and multi-line, multi-style text in text nodes.
  + Support for Hi-DPI displays has been added in this release.
  + The CSS Styleable\* classes became public API. See the [javafx.css](http://docs.oracle.com/javase/8/javafx/api/javafx/css/package-frame.html) javadoc for more information.
  + The new [ScheduledService](http://docs.oracle.com/javase/8/javafx/api/javafx/concurrent/ScheduledService.html) class allows to automatically restart the service.
  + JavaFX is now available for ARM platforms. JDK for ARM includes the base, graphics and controls components of JavaFX.
* [Tools](http://docs.oracle.com/javase/8/docs/technotes/tools/enhancements-8.html)
  + The jjs command is provided to invoke the Nashorn engine.
  + The java command launches JavaFX applications.
  + The java man page has been reworked.
  + The jdeps command-line tool is provided for analyzing class files.
  + Java Management Extensions (JMX) provide remote access to diagnostic commands.
  + The jarsigner tool has an option for requesting a signed time stamp from a Time Stamping Authority (TSA).
  + [Javac tool](http://docs.oracle.com/javase/8/docs/technotes/guides/javac/index.html)
    - The -parameters option of the javac command can be used to store formal parameter names and enable the Reflection API to retrieve formal parameter names.
    - The type rules for equality operators in the Java Language Specification (JLS) Section 15.21 are now correctly enforced by the javac command.
    - The javac tool now has support for checking the content of javadoc comments for issues that could lead to various problems, such as invalid HTML or accessibility issues, in the files that are generated when javadoc is run. The feature is enabled by the new -Xdoclint option. For more details, see the output from running "javac -X". This feature is also available in the javadoc tool, and is enabled there by default.
    - The javac tool now provides the ability to generate native headers, as needed. This removes the need to run the javah tool as a separate step in the build pipeline. The feature is enabled in javac by using the new -h option, which is used to specify a directory in which the header files should be written. Header files will be generated for any class which has either native methods, or constant fields annotated with a new annotation of type java.lang.annotation.Native.
  + [Javadoc tool](http://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/whatsnew-8.html)
    - The javadoc tool supports the new DocTree API that enables you to traverse Javadoc comments as abstract syntax trees.
    - The javadoc tool supports the new Javadoc Access API that enables you to invoke the Javadoc tool directly from a Java application, without executing a new process. See the [javadoc what's new](http://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/whatsnew-8.html) page for more information.
    - The javadoc tool now has support for checking the content of javadoc comments for issues that could lead to various problems, such as invalid HTML or accessibility issues, in the files that are generated when javadoc is run. The feature is enabled by default, and can also be controlled by the new -Xdoclint option. For more details, see the output from running "javadoc -X". This feature is also available in the javac tool, although it is not enabled by default there.
* [Internationalization](http://docs.oracle.com/javase/8/docs/technotes/guides/intl/enhancements.8.html)
  + Unicode Enhancements, including support for Unicode 6.2.0
  + Adoption of Unicode CLDR Data and the java.locale.providers System Property
  + New Calendar and Locale APIs
  + Ability to Install a Custom Resource Bundle as an Extension
* [Deployment](http://docs.oracle.com/javase/8/docs/technotes/guides/jweb/enhancements-8.html)
  + For sandbox applets and Java Web Start applications, URLPermission is now used to allow connections back to the server from which they were started. SocketPermission is no longer granted.
  + The Permissions attribute is required in the JAR file manifest of the main JAR file at all security levels.
* [Date-Time Package](http://docs.oracle.com/javase/8/docs/technotes/guides/datetime/index.html) - a new set of packages that provide a comprehensive date-time model.
* [Scripting](http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/enhancements.html#jdk8)
  + [Nashorn](http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/) Javascript Engine
* [Pack200](http://docs.oracle.com/javase/8/docs/technotes/guides/pack200/enhancements.html)
  + Pack200 Support for Constant Pool Entries and New Bytecodes Introduced by JSR 292
  + JDK8 support for class files changes specified by JSR-292, JSR-308 and JSR-335
* [IO and NIO](http://docs.oracle.com/javase/8/docs/technotes/guides/io/enhancements.html#jdk8)
  + New SelectorProvider implementation for Solaris based on the Solaris event port mechanism. To use, run with the system property java.nio.channels.spi.Selector set to the value sun.nio.ch.EventPortSelectorProvider.
  + Decrease in the size of the <JDK\_HOME>/jre/lib/charsets.jar file
  + Performance improvement for the java.lang.String(byte[], \*) constructor and the java.lang.String.getBytes() method.
* [java.lang and java.util Packages](http://docs.oracle.com/javase/8/docs/technotes/guides/lang/enhancements.html#jdk8)
  + Parallel Array Sorting
  + Standard Encoding and Decoding Base64
  + Unsigned Arithmetic Support
* [JDBC](http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/)
  + The JDBC-ODBC Bridge has been removed.
  + JDBC 4.2 introduces new features.
* Java DB
  + JDK 8 includes Java DB 10.10.
* [Networking](http://docs.oracle.com/javase/8/docs/technotes/guides/net/enhancements-8.0.html)
  + The class java.net.URLPermission has been added.
  + In the class java.net.HttpURLConnection, if a security manager is installed, calls that request to open a connection require permission.
* [Concurrency](http://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/changes8.html)
  + Classes and interfaces have been added to the java.util.concurrent package.
  + Methods have been added to the java.util.concurrent.ConcurrentHashMap class to support aggregate operations based on the newly added streams facility and lambda expressions.
  + Classes have been added to the java.util.concurrent.atomic package to support scalable updatable variables.
  + Methods have been added to the java.util.concurrent.ForkJoinPool class to support a common pool.
  + The java.util.concurrent.locks.StampedLock class has been added to provide a capability-based lock with three modes for controlling read/write access.
* [Java XML](http://docs.oracle.com/javase/8/docs/technotes/guides/xml/enhancements.html) - [JAXP](http://docs.oracle.com/javase/8/docs/technotes/guides/xml/jaxp/enhancements-8.html)
* [HotSpot](http://docs.oracle.com/javase/8/docs/technotes/guides/vm/)
  + Hardware intrinsics were added to use Advanced Encryption Standard (AES). The UseAES and UseAESIntrinsics flags are available to enable the hardware-based AES intrinsics for Intel hardware. The hardware must be 2010 or newer Westmere hardware. For example, to enable hardware AES, use the following flags:
  + -XX:+UseAES -XX:+UseAESIntrinsics

To disable hardware AES use the following flags:

-XX:-UseAES -XX:-UseAESIntrinsics

* + Removal of PermGen.
  + Default Methods in the Java Programming Language are supported by the byte code instructions for method invocation.
* [Java Mission Control 5.3 Release Notes](http://www.oracle.com/technetwork/java/javase/jmc53-release-notes-2157171.html)
  + JDK 8 includes Java Mission Control 5.3.