DESIGN (HTTPS://CODETE.COM/BLOG/CATEGORY/DESIGN/)
DEVELOPMENT (HTTPS://CODETE.COM/BLOG/CATEGORY/DEVELOPMENT/)
ENTERPRISE (HTTPS://CODETE.COM/BLOG/CATEGORY/ENTERPRISE/)
KNOW-HOW (HTTPS://CODETE.COM/BLOG/CATEGORY/KNOW-HOW/)
QUALITY (HTTPS://CODETE.COM/BLOG/CATEGORY/QUALITY/)
TECHNOLOGY (HTTPS://CODETE.COM/BLOG/CATEGORY/TECHNOLOGY/)

From Java 8 to Java 11 – Quick Guide

July 3, 2018

Preface

We are all familiar with Java 8 and its ingenious features, but over 4 years have passed from the day of Java 8 release and Java has made a few more big steps forward, with which not all of us had a chance to familiarize. And that's definitely high time to make up for it, because during this time we already had Java 9 release, which had been quickly superseded by Java 10 and there is Java 11 release just around the corner, which is going to be an important release, because similarly to Java 8, it will be a long-term-support version.

New Java release cycle

Oracle has announced that after releasing Java 9, they are changing the release cycle scheme to a much faster one. Instead of releasing only major increments twice a decade, we'll be getting Java updates twice a year and a LTS version every three years. That is a big shift and in our opinion a very good response to accusations that Java is not adapting to market needs fast enough.

That said, Java 9 was released in 2017 with immediate plan for being superseded by Java 10 in March 2018. Similarly Java 11 is planned to replace Java 10 in September 2018.

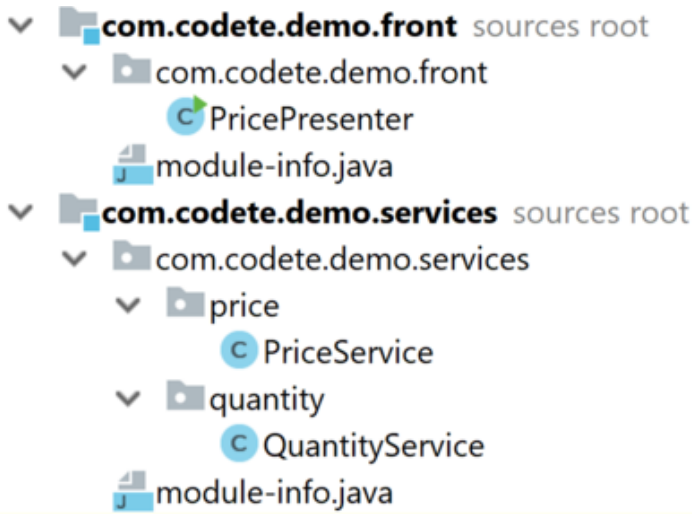
Java 9

Project Jigsaw

The flagship feature of Java 9 was Jigsaw project that introduced modularity to monolithic Java SE ecosystem. The primary goal of the Jigsaw project was to make the Java SE Platform and the JDK more easily scalable down to small computing devices.

Modularization of the JDK enables the source code to be completely restructured in order to make it easier to maintain. Module is a self-describing collection of code, data and resources. It consists of `module-info.java` file and one or more packages. Modules offer stronger encapsulation than jars.

Let's say that we have two modules in our application:



Each module has a `module-info.java` file that describes module's dependencies. Module `com.codete.demo.service` contains two packages: `price` and `quantity`. We want to expose only `price` package to third-party modules and keep `quantity` private. What we need to do is to define it in `module-info.java` file:

```
module com.codete.demo.services {
    exports com.codete.demo.services.price;
}
```

Class PricePresenter is defined in `com.codete.demo.front` module and requires `PriceService` defined in `com.codete.demo.service`, so we need to reflect it in `module-info.java`:

```
module com.codete.demo.front {
    requires  com.codete.demo.services;
}
```

JShell

Java 9 finally provided a Read-Eval-Print Loop (REPL) tool. This is an interactive programming tool which is continually reading user input, evaluating it and printing the output value or a description of the state change the input caused.

Let's declare a sample stream:

```
jshell> Stream<String> names = Stream.of("John", "George", "Susan","Tom");
names ==> java.util.stream.ReferencePipeline$Head@6e1567f1
```

Then create method that will print it for us:

```
jshell> public void printStringStream(Stream<String> input){ System.out.println(input.collect(
|  created method printStringStream(Stream<String>)
```

Now, invoke already declared method:

```
jshell> printStringStream(names);
[John, George, Susan, Tom]
```

Jshell supports Tab completion to type commands faster. For example put "names.co" in your Jshell command prompt and then press Tab key:
`jshell> names.co`
You will see all possible methods, that you can invoke on above stream, that starts with "co".

Except Java language expressions, Jshell provides it's own commands:
`jshell> /imports`
Will return list of imports. Please notice, that Jshell by default returns couple of them:

```
jshell> /imports
|    import java.io.*
|    import java.math.*
|    import java.net.*
|    import java.nio.file.*
|    import java.util.*
|    import java.util.concurrent.*
|    import java.util.function.*
|    import java.util.prefs.*
|    import java.util.regex.*
|    import java.util.stream.*
```

All of needed packages or classes that are not listed above, can be imported like in normal Java class.

Did you forget an already defined variable? That's not a problem! Just type:

```
jshell> /vars
|    Stream<String> names = java.util.stream.ReferencePipeline$Head@6e1567f1
```

The same way you can list all of declared methods:

```
jshell> /methods
|    void printStringStream(Stream<String>)
```

If you want close Jshell tool, just type:

```
jshell> /exit
|  Goodbye
```

Default garbage collector

Oracle proposed to make G1 garbage collector (which was first introduced in Java 8) the default one from Java 9. Switching to G1 provides better overall experience than a throughput-oriented collectors such as the Parallel GC, which was the default one in prior Java versions.

Language changes in Java SE 9

Private methods in interfaces

Starting from Java 9, we can define private methods in interfaces.

Let's say we have interface that returns number of available white and red items:

```
public interface ClientService {

    default long getNumberOfWhiteItems(List<Item> items) {
        return items.stream()
                    .filter(Item::isAvailable)
                    .filter(item -> item.getColor() == Color.WHITE)
                    .count();
    }

    default long getNumberOfRedItems(List<Item> items) {
        return items.stream()
                    .filter(Item::isAvailable)
                    .filter(item -> item.getColor() == Color.RED)
                    .count();
    }

}
```

We can extract common business logic to private methods and make them cleaner:

```
public interface ClientService {

    private long getNumberOfAvailableItems(List<Item> items, Predicate<Item> filteringPredicat
        return items.stream()
                    .filter(Item::isAvailable)
                    .filter(filteringPredicate)
                    .count();
    }

    default long getNumberOfWhiteItems(List<Item> items) {
        return getNumberOfAvailableItems(items, item -> item.getColor() == Color.WHITE);
    }

    default long getNumberOfRedItems(List<Item> items) {
        return getNumberOfAvailableItems(items, item -> item.getColor() == Color.RED);
    }

}
```

Thanks to this we gain more flexibility in exposing only intended methods to API users and can avoid code duplication in default methods.

Optional Class

Java 9 comes with new methods in Optional class.

Optional.ifPresentOrElse()

Let's assume that we have two methods to display information about shopping basket depending on its presence.

```
Optional<ShoppingBasket> shoppingBasket = findShoppingBasket();
```

Instead of writing traditional if/else construction

```
if (shoppingBasket.isPresent()) {
   displayShoppingBasketContent(shoppingBasket.get());
} else {
   displayEmptyShoppingBasket();
}
```

We can use `ifPresentOrElse` method:

```
shoppingBasket.ifPresentOrElse(this::displayShoppingBasketContent, this::displayEmptyShoppingBa
```

Optional.or()

Other new method available in Optional class is `or`. If value is present, method will return Optional describing the value, otherwise returns an Optional produced by the supplying function:

```
Optional<ShoppingBasket> shoppingBasket = findShoppingBasket().or(() -> Optional.of(new Shoppi
```

Optional.stream()

Since Java 9 we can treat the Optional instance as a Stream and utilize all methods from Stream API. This makes it also much more convenient to work with streams of Optionals.

Immutable Collection Factory

Java 9 came with static methods on the List, Set and Map interfaces for creating unmodifiable instances of those collections. Till now, to create immutable collection, programmers had to construct it, store it in a local variable, fill with `add` method and wrap into a unmodifiable collection.

```
private Set<String>  createImmutableSet() {
   Set<String> names = new HashSet<>();
   names.add("John");
   names.add("George");
   names.add("Betty");

   return Collections.unmodifiableSet(names);
}
```

Now, you can replace it with:

```
private Set<String>  createImmutableSet() {
   return Set.of("John", "George", "Betty");
}
```

It's also trivial to create an immutable map:

```
return Map.of("John", 1, "Betty", 2);
```

CompletableFuture improvements

The newest version of Java introduces several new methods to CompletableFuture class. Most significant one relates to timeouts.

`orTimeout` exceptionally completes a ComplatableFuture if it's not completed in given time:

```
CompletableFuture<String> newOne = future.orTimeout(10, TimeUnit.SECONDS);
```

The other one is `completeOnTimeout` :

```
CompletableFuture<String> newOne = future.completeOnTimeout("value in case of timeout", 10, Tir
```

The method completes current CompletableFuture with "value in case of timeout" if not completed before the timeout.

Enhancements in @Deprecated

Java 9 provides enhancements for @Deprecated annotation by adding two new parameters to it:

`since` – defines version in which the annotated element became deprecated

`forRemoval` – indicates whether the annotated element is subject to removal in future version

```
@Deprecated(since = "4", forRemoval = true)
public String getDescription()
```

Java 10

Java 10 didn't come with groundbreaking features, however, it was delivered as promised, only 6 months after Java 9 release and we need to commend that, because in the past years there were always problems with delivering promised Java updates on time.

Keyword "var" (Local-Variable Type Inference)

Probably the most recognisable feature of this release (at least from developer's point of view). The `var` keyword that has been introduced can replace strict variable type declaration if the type may be easily inferred by the compiler. Therefore we can type less, don't need to duplicate type information and it just makes the code look nicer.
We can use `var` in the context of local variables (instantly initialized), `for` loops and `try-with-resources` blocks.

We can't use `var` e.g. when declaring class fields or method parameters.

A few examples of correct usage of `var` :

```
var company = "Codete"; // infers String
var characters = company.toCharArray(); // infers char[]
```

```
var numbers = List.of("a", "b", "c");
for (var nr : numbers) {
   System.out.print(nr + " ");
}
```

And a few code samples with var which will not compile:

```
var foo;
var ints = {0, 1, 2};
var appendSpace = a -> a + " ";
```

```
private var getFoo() {
   return "foo";
}
```

Also it's important that the compiler will resolve the exact type of the object that a variable is initialized with. Therefore we will get a compilation error in the following situation:

```
var users = new ArrayList<User>(); // type resolved: ArrayList<User>
users = new LinkedList<User>(); // error
```

Additions to JDK

A few small additions came also to the standard class library. For instance, there is a new overloaded version of `Optional.orElseThrow()` which doesn't take any parameters and throws `NoSuchElementException` by default.

Additionally, API for creating unmodifiable collections has been improved by adding `List.copyOf()`, `Set.copyOf()`, `Map.copyOf()` methods, which are factory methods for creating unmodifiable instances from existing collections. Also, `Collectors` class has some new methods like `toUnmodifiableList`, `toUnmodifiableSet`, `toUnmodifiableMap`.

Parallel Full GC for G1

As we mentioned before, G1 is the default GC starting from Java 9. It was mainly designed to avoid full collections and preventing the "stop the world" event. It increased the performance significantly, however, there was still probability that if concurrent collections couldn't reclaim memory quickly enough, then classic full GC would be used as a fallback.

Therefore in Java 10 they took care of this particular situation too and improved the algorithm to parallelize the full GC.

Java 11

Local-Variable Syntax for Lambda Parameters

Currently when we want to specify lambda parameters types, we need to do this explicitly and can't use var keyword introduced in Java 10.

```
IntFunction<Integer> doubleIt1 = (int x) -> x * 2; // Correct
IntFunction<Integer> doubleIt2 = (var x) -> x * 2; // Error
```

The second line won't compile in Java 10, but will compile in Java 11.
But, hey, why do we need that at all? If we can write it like this:

```
IntFunction<Integer> doubleIt = x -> x * 2;
```

even in Java 8 and type inference will do the trick.
Yes, it's true, but sometimes you have to use explicit typing in lambda and then var may be useful, e.g. when you want to make a parameter final or add annotations.

```
IntFunction<Integer> doubleIt1 = (@Valid final int x) -> x * 2;   // Compiles since Java 8
IntFunction<Integer> doubleIt2 = (@Valid final var x) -> x * 2;   // Will compile from Java 11
```

Launch Single-File Source-Code Programs

With that feature simple, single-file programs (which are common especially at early stages of learning Java) won't need to be compiled separately. Typing simply

```
java SimpleProgram.java
```

will run the program immediately.

More than just features

Currently, list of planned features for Java 11 is quite small, but release is planned for September 2018 and perhaps some extra features will be added up to that time.

However, these are not fancy features what is going to make Java 11 release so important, but the fact that this version will be the LTS version and thus it will be the most reliable substitute for popular Java 8.

Conclusion

The change of Java release cycle, many syntax-sugar features that came in Java 9 & 10, plenty of safety & performance improvements and the approaching Java 11 platform long term support bring the conclusion that Oracle has a vision for adopting innovation to the language and maintains high level of stability at the same time. This all makes Java development even more interesting and we're sure that there are more great news coming our way.

This article presented a subset of features introduced to Java between versions 9 and 11, which in our subjective vision are the most interesting and useful ones from Java developer's perspective. Hope you enjoyed it.

Keywords:

Java 9, JDK 9, Java 10, JDK 10, Java 11, JDK 11, var, local-variable, Jigsaw, modularity, Jshell, REPL, GC, G1, Garbage Collector, Private method, Interfaces, Immutable, collections, List, Set, Map, CompletableFuture, timeout, deprecated

Sources:

http://openjdk.java.net (http://openjdk.java.net)

http://www.journaldev.com/13106/javase9-module-system-part1 (http://www.journaldev.com/13106/javase9-module-system-part1)

https://blogs.oracle.com/sundararajan/entry/playing_with_java_java9_repl (https://blogs.oracle.com/sundararajan/entry/playing_with_java_java9_repl)

http://www.baeldung.com/new-java-9 (http://www.baeldung.com/new-java-9)

https://bentolor.github.io/java9-in-action/#/5/1 (https://bentolor.github.io/java9-in-action/#/5/1)

https://blog.takipi.com/java-11-will-include-more-than-just-features/ (https://blog.takipi.com/java-11-will-include-more-than-just-features/)

ALEKSANDER SOBOL (HTTPS://CODETE.COM/BLOG/AUTH...

Software engineer at Codete with over 6 years of experience in professional software development. Caring about high quality and clean code. Recently focused on blockchain technologies

SHARE:

RELATED POSTS

EVENT (HTTPS://CODETE.COM/BLOG/CATEGORY/EVENT/)
KNOW-HOW (HTTPS://CODETE.COM/BLOG/CATEGORY/KNOW-HOW/)
PERSONAL DEVELOPMENT (HTTPS://CODETE.COM/BLOG/CATEGORY/PERSONAL-DEVELOPMENT/)

CodeteCON #KRK5 – technology event organized by Codete – 15/11/19 (https://codete.com/blog/codetecon-krk5-technology-event-organized-by-codete-15-11-19/)

November 26, 2019

KNOW-HOW (HTTPS://CODETE.COM/BLOG/CATEGORY/KNOW-HOW/)
TECHNOLOGY (HTTPS://CODETE.COM/BLOG/CATEGORY/TECHNOLOGY/)

5 New Recruitment Technology Trends You Should Implement in Your Hiring Process (https://codete.com/blog/5-new-recruitment-technology-trends-you-should-implement-in-your-hiring-process/)

November 19, 2019

CONSULTING (HTTPS://CODETE.COM/BLOG/CATEGORY/CONSULTING/)
ENTERPRISE (HTTPS://CODETE.COM/BLOG/CATEGORY/ENTERPRISE/)
TECHNOLOGY (HTTPS://CODETE.COM/BLOG/CATEGORY/TECHNOLOGY/)

Business Software Consulting – Codete Enterprise Software House (https://codete.com/blog/business-software-consulting-codete-enterprise-software-house/)

November 12, 2019

SUBSCRIBE

Sign up for our Newsletter and keep up to date.

E-MAIL

CONTINUOUSLY
DELIVERING
TECHNOLOGY

GET IN TOUCH
mail@codete.com (mailto:mail@codete.com)