JAVA {MAGAZINE

Onafhankelijk tijdschrift voor de Java-professional

.nl.jug

02 > 2024

NO MORE MISTAKES

Automating large-scale refactoring's with Error Prone

› JAVA 22
WHAT'S NEW?

› DIVIDE AND CONQUER
DATABASE TABLE PARTITIONING

› PASSKEYS
A PASSWORDLESS FUTURE

Become a NLJUG member!

NLJUG Base, Core, and Key memberships giving members privilege for different events each year. Such as Free access to J-Spring, J-Fall and TEQnation. And many more benefits inside the JAVA community.

NLJUG is a Java community dedicated to developing and making the platform accessible for knowledge sharing and networking. With over 4500 members and 80 businesspartners.

BECOME A CORE MEMBER

€ 125,–

- Priority access to the J-Fall conference Food and drinks included
- Free Java Magazine subscription (4 issues a year)
- NLJUG newsletter containing the most relevant knowledge and meetups
- 50% discounted access to the J-Spring conference
- Free access to NLJUG University sessions
- 50% discounted acces to J-Fall Pre-con Masterclasses
- Additional exclusive discounts on courses and training organized by our partners
- 50% discounted acces to TEQnation conference

BECOME A KEY MEMBER

€ 279,–

- VIP access to the J-Fall conference Food and drinks included
- Free Java Magazine subscription (4 issues a year)
- NLJUG newsletter containing the most relevant knowledge and meetups
- Free access to J-Spring conference (worth € 169,-)
- Free access to NLJUG University sessions
- Free access to J-Fall Pre-con Masterclasses (worth € 129,-)
- Additional exclusive discounts on courses and training organized by our partners
- Access to J-Spring and J-Fall VIP Lounges incl. reserved free parking
- Exclusive access to the NLJUG Core Slack channel, where you can discuss the future of our community
- NLJUG Key Member Swag (including NLJUG Core Member Hoody, Core Member pin and much more)
- NLJUG Key Member visible on conference badges
- Free acces to TEQnation conference (worth € 239,-)

Advantages of becoming an NLJUG member:

- Free access to J-Fall
- Free access to the NLJUG University sessions
- 4 times a year receive the (printed) Java Magazine
- Discounts on partner events

nl. jug

https://nljug.org/lidworden/

{ COLOFON } JAVA MAGAZINE 02-2024

Project coördinatie:
Lieke Stomps

Communitymanager:
Simon de Groot

Eindredactie:
Suzanne Muller

Auteurs:
Felienne Hermans, Brian Vermeer, Bert Breeman, Julien Lengrand-Lambert, Maja Reissner, Ivo Woltring, Hanno Embregts

Redactiecommissie:
Hanno Embregts, Ivo Woltring, Thomas Zeeman, Laurens van der Kooi, Rogier Pijpers, Georgina Fábián, Deepu K Sasidharan, Rick Ossendrijver, Maja Reißner en Brian Vermeer

Vormgeving:
Wonderworks, Haarlem

Traffic & Media order:
Marco Verhoog

Drukkerij:
Senefelder Misset, Doetinchem

Advertenties:
Richelle Bussenius
E-mail: richelle.bussenius@nljug.org
Telefoon: 023 752 39 22
Fax: 023 535 96 27
Marc Post
E-mail: mpost@reshift.nl
Telefoon: 023 543 00 08

Abonnementenadministratie:
Tanja Ekel

De prijs van een lidmaatschap van de NLJUG verschilt per membership tier die u afneemt. Een reguliere "base" membership van de NLJUG kost € 59,50 per jaar, waarbij Java Magazine gratis verschijnt. Naast het Java Magazine krijgt u gratis toegang tot de vele NLJUG workshops en het J-Fall congres. Het NLJUG is lid van het wereldwijde netwerk van JAVA user groups. Voor meer informatie of wilt u lid worden, **zie www.nljug.org**. Een nieuw lidmaatschap wordt gestart met de eerst mogelijke editie voor een bepaalde duur. Het lidmaatschap zal na de eerste (betalings)periode stilzwijgend worden omgezet naar lidmaatschap van onbepaalde duur, tenzij u uiterlijk één maand voor afloop van het initiële lidmaatschap schriftelijk (per brief of mail) opzegt. Na de omzetting voor onbepaalde duur kan op ieder moment schriftelijk worden opgezegd per wettelijk voorgeschreven termijn van 3 maanden.
Een lidmaatschap is alleen mogelijk in Nederland en België.
Uw opzegging ontvangen wij bij voorkeur telefonisch. U kunt de Klantenservice bereiken via: 023-5364401. Verder kunt u mailen naar **members@nljug.org** of schrijven naar NLJUG BV, Ledenadministratie, Nijverheidsweg 18, 2031 CP Haarlem.
Verhuisberichten of bezorgklachten kunt u doorgeven via **members@nljug.org** (Klantenservice).

Wij nemen je gegevens, zoals naam, adres en telefoonnummer op in een gegevensbestand. De verwerking van uw gegevens voeren wij uit conform de bepalingen in de Algemene Verordening Gegevensbescherming. De gegevens worden gebruikt voor de uitvoering van afgesloten overeenkomsten, zoals de abonnementenadministratie en, indien je daar toestemming voor hebt gegeven, om je op de hoogte te houden van interessante informatie en/of aanbiedingen. Je kunt uw persoonsgegevens opvragen om inzicht te krijgen in welke gegevens wij van je hebben, deze te corrigeren of, na beëindiging van de abonnee-overeenkomst, te laten verwijderen. Stuur hiertoe een kaartje aan NLJUG BV, afd. klantenservice, Nijverheidsweg 18, 2031 CP Haarlem of een e-mail naar **members@nljug.org**.

.nl.
jug

VOORWOORD

Lente

April showers bring May flowers. Terwijl de lente zich ontvouwt, de wereld tot leven komt en TEQnation (22 mei) en J-spring (13 juni) om de hoek komen kijken, is het wederom gelukt om de tofste artikelen te verzamelen voor deze editie van het Java magazine! Lees meer over de opkomende evenementen in de vooruitblik vanaf pagina 22.

V

Lieke Stomps
Projectcoördinator NLJUG
info@nljug.org

INHOUD

Database table partitioning

6 Many projects go live with a datastore, most of them using a relational database. At the start of a project, we may not yet know if and how successful an application might become or how it may grow beyond the original scope. Thus we may not pull out all the stops on optimising the database structure for e.g. longevity.

Kubernetes Operator

15 Digipoort wordt herbouwd volgens het event-driven architectuurmodel, met Kubernetes als container platform. Digipoort kent een complexe infrastructuur met veel configuraties en een applicatie landschap dat dynamisch verandert. Om dit beheersbaar te maken is er een Kubernetes Operator geïmplementeerd, die het grootste deel van dit werk automatiseert. Het doel van dit artikel is om de opgedane kennis te delen en om te laten zien dat het implementeren van een eigen Operator eigenlijk heel toegankelijk is.

Error Prone introduction

34 When it comes to writing good computer programs, avoiding mistakes is key. To prevent errors, several processes are generally put in place, such as thorough review procedures and the use of static analysis tools. However, despite these measures, human error remains a stubborn obstacle in software development. Issues frequently slip through the review process, and tools like Checkstyle, SonarCloud, and SpotBugs only highlight some common problems. This leaves the developer with the task of manually applying the suggested improvements.

SCHRIJVEN VOOR JAVA MAGAZINE?

Ben je een enthousiast lid van NLJUG en zou je graag willen bijdragen aan Java Magazine? Of ben je werkzaam in de IT en zou je vanuit je functie graag je kennis willen delen met de NLJUG-community? Dat kan! Neem contact op met de redactie, leg uit op welk gebied je expertise ligt en over welk onderwerp je graag zou willen schrijven. Direct artikelen inleveren mag ook. Mail naar info@nljug.org en wij nemen zo spoedig mogelijk contact met je op.

10 Fun with Functional Programming – crafting your own Monad (part 2)

20 Innovation awards

21 Byte Size

22 J - Spring vooruitblik

25 TEQnation vooruitblik

30 Java 22

39 Passkeys for Java Developers

44 De ethiek van een ethische hacker

46 Bestuurscolumn

47 Masters of Java

TABLE PARTINIONING
Divide and Conquer

Many projects go live with a datastore, most of them using a relational database. At the start of a project, we may not yet know if and how successful an application might become or how it may grow beyond the original scope. Thus we may not pull out all the stops on optimising the database structure for e.g. longevity.

Thomas Zeeman is a software architect at Trifork. He works on projects both small and large and has many interests, including how to keep software maintainable.

One such thing that can happen in case your application continues to thrive is that some tables grow and grow and grow... At some point you may want to consider your options to keep these tables manageable.

One such option is to partition such a table. This means the table will be cut into pieces based on a property in the table, called the partition key. A property that does need to be chosen carefully, as it should help spread the data somewhat evenly over the partitions and should, ideally, be present in every query on that table.

{ THE THEORY... }

In principle, partitioning of a table means that we set up our table such that it looks like it is a single table, but in practice it consists of several smaller tables (see image 1). Each with a particular partition of the data. This could be data based on a particular property having a specific value or because it fits in a defined range of values. The former can i.e. be a string or number, the latter is typically a range of numbers or dates, but many databases also support hashes. It would be ideal if that property is also the primary key of the table, but that's not always the case.

So, when should we consider partitioning? A couple of reasons mentioned in the various database manuals include

- When a table reaches a certain size; i.e. Oracle mentions 2 GB [1]
- When your table contains historical data and you add more data as time goes by; i.e. a billing system where you create new invoices over time
- When your data has different storage requirements based on its state; is it an active account or not, does it have to be searchable online or can it go into archival storage?
- If you want to improve performance... possibly [2]
- Data retention, when your data can, or should, be removed after some time

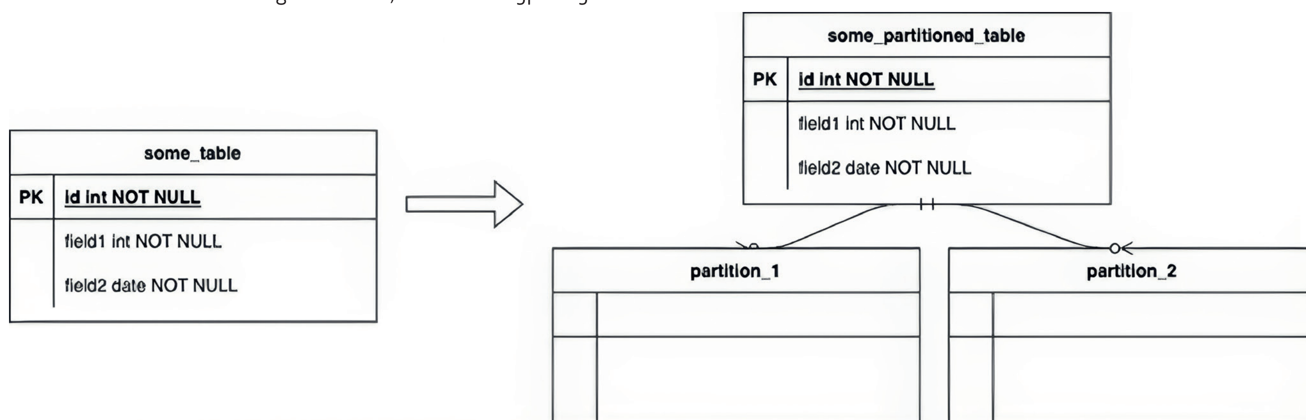


Image 1

{ ...AND THE PRACTICE }

All of the major databases support partitioning of tables in some form. All of them have their own limitations on how many partitions they support, what data types are available as partition keys, whether indexes work across all partitions, and whether they can then also guarantee unique constraints across the entire table or only per partition. To top it off, it can also differ per version of a particular database. So, for more information on those nitty, gritty details I'll refer to the individual database reference documentation. In this article I'll use PostgreSQL to illustrate some examples.

{ SCENARIO 1: MULTI-TENANCY }

So let's make this a bit more concrete and look at two scenarios. First an application where you store data for multiple customers in a way that can be described as multi-tenant. A single application with data for multiple users that should not be mixed up at all. Sure, we could start with a completely isolated application per tenant, but we all know that that may not be affordable when you start off, or even considered when it's built for the first customer. So, we have a set of tables, each with a column tenant. That makes it an ideal candidate for the partition key.

L1

```
ALTER TABLE invoices RENAME TO invoices_legacy;
CREATE TABLE invoices (
                     id                     BIGINT          NOT NULL,
                     invoice_date           DATE            NOT NULL,
                     description            VARCHAR(255)    NOT NULL,
                     amount                 BIGINT          NOT NULL,
                     discount               BIGINT          DEFAULT 0
) PARTITION BY RANGE (invoice_date);
-- use an arbitrary old date to fill the FROM date. TO is exclusive.
CREATE TABLE invoices_2023 PARTITION OF invoices
   FOR VALUES FROM ('1900-01-01'::DATE) TO ('2024-01-01'::DATE);
CREATE TABLE invoices_2024 PARTITION OF invoices
   FOR VALUES FROM ('2024-01-01'::DATE) TO ('2025-01-01'::DATE);

INSERT INTO invoices (id, invoice_date, description, amount, discount)
SELECT (id, invoice_date, description, amount, discount) FROM invoices_legacy;
```

L2

```
-- ensure we do not check on the validity of the data, because we already know it's safe
ALTER TABLE invoices ADD CONSTRAINT invoices_y2023_c
   CHECK(invoice_date >= '1900-01-01'::DATE AND invoice_date < '2024-01-01'::DATE) NOT VALID;
ALTER TABLE invoices ATTACH PARTITION invoices_legacy
   FOR VALUES FROM ('1900-01-01'::DATE) TO ('2024-01-01'::DATE);
-- now that we have attached the data, we can remove the temporary constraint
ALTER TABLE invoices_legacy DROP CONSTRAINT invoices_y2023_c;
```

Some of the benefits of partitioning the tables in this scenario are that you can isolate all the data for a particular tenant, maintenance on data for a particular tenant becomes possible without disturbing others, and, if needs be, it's also easier to migrate a tenant to a separate instance if it grows too big to handle or has additional requirements due to changing rules and regulations like GDPR.

{ SCENARIO 2: HISTORICAL DATA }

Another example would be historical data. Many businesses will have some kind of billing or financial records system and they typically have to keep those records for a very long time due to tax rules. That does not mean you have to keep them easily available all the time though, and at some point you may also be able to move them to cheaper, archival storage or even throw them away.

The obvious benefit is of course that you can keep your active records set limited and thus keep performance in check, while not having to throw away or have more expensive data migration actions in place to get the same result. It also becomes easier to drop older data if it can finally be shredded. Just detach the partition, if you haven't done so already, and then drop that single table instead of executing a series of delete statements followed by an autovacuum of the table to make up for the freed up space. The detach is trivial and almost instantaneous, the delete and subsequent re-index and vacuum are more complex and can be very taxing.

{ IMPLEMENTING IT IN YOUR DATABASE }

If you have by now decided to go this route, what do you have to do to get here then? The easy approach would be to rename the old table, then create the new table and copy over all the data as is done in Listing 1.

This is not taking into account any indexes created previously for the old table, queries that do not take the invoice_date into account or views on the old table that may now point to the renamed table or other things your database of choice may or may not do due to a rename. Oh, and also not taking into account whether or not the choice of partition key here is actually the best possible… that I leave as an exercise for the reader ;-)

Depending on the amount of data in the original table, this may also take quite some time to execute. In one case I was involved with, the original approach the developers took, the timer got to about a day for a dataset similar to production. Needless to say, at

```
@Entity
public class Invoices {
    @Id
    private Long id;
    @PartitionKey
    private LocalDate invoiceDate;
    …
}
```
L3

this point we asked a database administrator to help with optimising this. The resulting script still started with a rename but didn't have to copy over the old data. So, instead of creating the new partition for the old data, we used a temporary constraint to prevent the database from checking all the data to see if it matches the range set on the partition. Listing 2 show how we could do this in our example:

You may also want to check your database documentation as some also have (limited) support to partition an existing table in place, i.e. MySQL [3].

Now you have some idea how to implement it in your database. That does not mean you're done with it yet. Depending on the type of the partition key, you will have to put some procedures in place to create new partitions or risk a grinding halt of your application as it tries to insert data into something that isn't there. Also detaching or even removing older partitions should be thought about. This could be automated but is not necessarily the best approach to take.

{ IMPLEMENTING IT IN YOUR APPLICATION? }

While table partitioning is primarily something in your database, it is not something you as an application developer can completely ignore. First and foremost, the choice of the partition key can be of effect on your queries to the database. If you do not take that into consideration, the database may have to query all partitions. That could lead to serious performance degradation.

Beyond that, you may be hit by some of the limitations of your database of choice when it comes to unique constraints (i.e. MySQL and Postgres have this) or the number of significant characters in names for tables, indexes and constraints (not too long ago limits of less than 64 characters were quite common, sometimes much less than that).

For those that use SQL queries directly, i.e. because you use JDBC or MyBatis, you're on your own to go over all the queries in your application that use this table and rewrite your queries to use the partition key if and where necessary.

If you use JPA you may have a little less to do. Hibernate has added some basic support in version 6.2 [4] which allows you to mark the @PartitionKey such that it will try to take that into account for at least update and delete statements (see Listing 3). Custom queries written in JPQL do still require attention from a developer to optimise.

EclipseLink also has a series of annotations for partitioning [5] but those target database partitioning or clustering. Also interesting and it can benefit from table partitioning, but not directly useful here.

{ ABOUT THAT PERFORMANCE? }

As one of the benefits, performance was mentioned but with a caveat. That is for a reason. Partitioning can indeed get you a nice performance boost, but only if your queries hit a single or a few partitions at most. Then it will only need to run over one or a few smaller indexes or smaller tables and that saves memory and going back-and-forth to the disk. So, if you get a performance improvement heavily depends on whether you can get your queries to include the partition key and thus get the query planner to limit the amount of data you have to search through.

{ SUMMARY }

I hope you got an idea on what you could do within your existing database when your tables grow beyond what you may initially have expected. Or maybe you already expect them to grow this large and you decide to start with this approach from the get go.

For some additional fun reading on an approach I have used on an existing set of tables, where we had constraints regarding existing data, possible downtime and some more, I leave you with this approach where we included the date in the id column of the table [6]. It has its pros and cons and required some customization in the code as well to make Hibernate aware of it, but it has been serving us for some time now. ‹

{ REFERENCES }

1 https://docs.oracle.com/en/database/oracle/oracle-database/19/vldbg/partition-concepts.html
2 https://learn.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes
3 https://dev.mysql.com/doc/refman/8.0/en/alter-table-partition-operations.html
4 https://in.relation.to/2023/02/08/hibernate-orm-62-partitioning/
5 https://wiki.eclipse.org/EclipseLink/Examples/JPA/Partitioning
6 https://database.one/blog/date_in_sequence_updates

CRAFTING A CUSTOM MONAD-LIKE TYPE IN JAVA Part 2

The first part of this article navigated the conceptual terrain of crafting the Conditional. Now, we delve into its practical implementation, exploring key functional programming concepts along the way.

Laurens van der Kooi is a Java Developer at Ordina JTech. He enjoys sharing knowledge and is a big fan of the Spring Framework and Functional Programming.

{ THE CONDITIONAL: A RECAP }

To recap the first article, the Conditional is a monad-like type that is able to organize conditional actions as a functional alternative to the traditional if-then-else-statements (Listing 1). We implemented the initial unit tests to facilitate Test-Driven Development (Listing 2).

L1

```
public static BigDecimal calculateNumber(int
number) {
   return Conditional.of(number)
      .firstMatching(
         applyIf(isEven(), times(2.5)),
         applyIf(isLargerThan(100), times(0.5))
         )
      .map(BigDecimal::valueOf)
      .orElse(BigDecimal.ZERO);
}

private static Function<Integer, Double>
times(double number) {
      return i -> i * number;
}

private static Predicate<Integer> isEven() {
      return i -> i % 2 == 0;
}

private static Predicate<Integer>
isLargerThan(int number) {
      return i -> i > number;
}
```

components of the functional pipelines under test:

- The instance fields of the Conditional and the of-constructor;
- The intermediate operation *firstMatching* along with its companion *applyIf* for syntactic sugar;
- The terminal operation orElse.

{ IMMUTABILITY }

Before we delve into implementing the Conditional, let's touch on a fundamental concept in functional programming: immutability. It emphasizes that once a data structure is created, it should remain unaltered. In functional programming, modifications create new, modified copies instead of altering the existing structure. This practice ensures data immutability, preventing unexpected changes, and enhances code reliability and readability.

To achieve immutability within a class, several guidelines should be followed:

- Declare all class members as final, making them not reassignable after initialization;
- Aim for immutable members. If not possible, ensure getters return copies of members instead of original references;
- Declare the class itself as final to prevent subclassing and potential introduction of mutable behavior through inheritance.

By adhering to these principles, a class becomes immutable, ensuring thread safety, improving debuggability, and contributing to a more robust and maintainable solution.

```
@Test
@DisplayName("a condition matches -> apply
   matching function")
void
conditionalWithOneConditionThatEvaluatesToTrue()
{
   var outcome =
   Conditional.of(2)
   .firstMatching(applyIf(isEven(), times(2)))
   .orElse(0.00);

   assertThat(outcome).isEqualTo(4.00);
}

@Test
@DisplayName("no condition matches -> return
    default value")
void conditionalWithOneConditionThatEvaluates-
   ToFalse() {
 var outcome =
 Conditional.of(3)
 .firstMatching(applyIf(isEven(), times(2)))
 .orElse(0.00);

 assertThat(outcome).isEqualTo(0.00);
}
```

{ DEFINING INSTANCE FIELDS AND CONSTRUCTORS }

Having examined the initial setup of the Conditional, we can identify several characteristics related to the instance fields and instantiation of this class:

- The Conditional has the ability to receive condition/function pairs;
- The evaluation of condition/function pairs takes place in the order they were supplied to the Conditional;
- The Conditional features a static constructor: *of*.

Let's introduce a specialized entity designed to encapsulate a condition/function pair (Listing 3). For this we introduce a generic Record type named *ConditionalAction*, intended to contain a condition (a *Predicate*) and an action (a *Function*). To ensure the integrity of our pair we implement a compact constructor that contains null checks for both fields. Note that *ConditionalAction* is an inner (record) class of Conditional since it has only meaning within that context.

The Conditional evaluates condition/function pairs in the order provided, requiring an ordered collection. Java's List, part of the *Collections Framework*, serves this purpose, providing various ways to instantiate immutable lists as well.

```
public record ConditionalAction<S, T>
   (Predicate<S> condition, Function<S, T>
   action) {
 public ConditionalAction {
 Objects.requireNonNull(condition);
 Objects.requireNonNull(action);
 }
}
```

Another aspect to consider regarding instance fields and the instantiation of the Conditional is the static constructor, named of, which is specifically designed to encapsulate the value intended to be wrapped by this monadic type. As emphasized in the first article, the type of this value is denoted as *S*.

To encapsulate the mentioned fields in an immutable manner, we mark them as private final. To enforce that developers consistently construct a Conditional by invoking the static of-constructor, we make the primary constructor private. Additionally, to prevent extension of the class, we declare Conditional as final. The implementation of the constructors and instance fields is illustrated in Listing 4. Please note that in the *of*-constructor we initially instantiate the Conditional with an `Collections.emptyList()`; this creates an immutable empty list.

```
public final class Conditional<S, T> {
private final S value;
private final List<ConditionalAction<S, T>>
   conditionalActions;

private Conditional(S value,
List<ConditionalAction<S, T>> actions) {
this.value = value;
this.conditionalActions = actions;
}

public static <S> Conditional<S, S> of(S value) {
 return new Conditional<>(value, Collections.
   emptyList());
}

 // ... other class members
}
```

{ IMPLEMENTING INTERMEDIATE OPERATIONS }

The next phase involves implementing a crucial element of the functional pipeline: the *firstMatching* method. To facilitate the construction of functional pipelines through method chaining, we

need to ensure that intermediate operations consistently return a new Conditional object that encapsulates the updated state resulting from the intermediate operation. This way we adhere to the principles of immutability throughout the execution of operations on the Conditional monad.

Having explored the first setup of the Conditional (Listing 1), it becomes apparent that methods like firstMatching and map yield a Conditional that potentially returns a value type different from the one in the initial Conditional of the functional pipeline.

Image 1 shows that our first setup starts as a `Conditional<Integer, Integer>`, at that point being a Conditional that takes an *Integer* and potentially returns an *Integer* as intended when we implemented the of-constructor. After supplying the *times* operation to *firstMatching*, we observe that the method returns a Conditional capable of returning a *Double*. This outcome arises because the *times* method returns a `Function<Integer, Double>`; a function that accepts an Integer and produces a *Double*.

Furthermore, the map method, when paired with `BigDecimal::valueOf`, produces a Conditional capable of returning a *BigDecimal*. Consequently, operations such as *firstMatching* and *map* can be regarded as transformations, facilitating the object's transformation within the monadic type. Later on, we'll explore a similar method: *flatMap*.

```
public static BigDecimal calculateNumber(int number) {
    return Conditional.of(number) Conditional<Integer, Integer>
            .firstMatching(
                    applyIf(isEven(), times( number: 2.5)),
                    applyIf(isLargerThan( number: 100), times( number: 0.5))
            ) Conditional<Integer, Double>
            .map(BigDecimal::valueOf) Conditional<Integer, BigDecimal>
            .orElse(BigDecimal.ZERO);
}
```

Image 1

{ IMPLEMENTING FIRSTMATCHING AND APPLYIF }

The *firstMatching* method is designed to accommodate an unspecified number of condition/function pairs. To enhance the readability of the pipeline, we'll introduce a method serving as syntactic sugar: *applyIf*.

In order to introduce flexibility, allowing the *firstMatching* method to return a Conditional capable of yielding a type different from the one from which the method was invoked, we need to introduce a new type, denoted as *U*. This signals to the compiler that a new generic type is involved. While the first generic type (S) remains unchanged to represent the type of value that is wrapped by the Conditional, the return type becomes `Conditional<S, U>`. With the method accepting an unspecified number of condition/function pairs, a varargs parameter for *ConditionalActions* is included. Listing 5 illustrates the implementation, emphasizing the creation of a new Conditional by using the wrapped value (contained in the value field) of the current instance, along with the provided ConditionalActions.

Since firstMatching accepts a varargs of *ConditionalAction*, *applyIf* logically returns a `ConditionalAction<S, U>` (Listing 6). Since it is intended to be a helper method, not acting upon the state of the Conditional, we mark this method as static.

{ IMPLEMENTING TERMINAL OPERATIONS }

Terminal operations, as discussed previously, initiate pipeline execution, evaluating each condition in the conditionalActions field of the Conditional. This evaluation halts upon finding the first matching condition, emphasizing the goal of identifying the initial match. Subsequently, the action from this first matching *ConditionalAction* is applied to the wrapped value.

To achieve this, we create a method (Listing 7) that streams the conditionalActions field, filters matching conditions, and returns an Optional containing the first identified action (a *Function*). This method can be used in all three terminal operations (*orElse*, *orElseGet*, and *orElseThrow*) to check for matching conditions.

L5

```
public <U> Conditional<S, U> firstMatching(
 ConditionalAction<S, U>... actions) {
 var actionsAsList = Arrays.stream(actions).
   toList();

 return new Conditional<>(value, actionsAsList);
}
```

L6

```
public static <S, U> ConditionalAction<S, U>
applyIf(Predicate<S> condition, Function<S, U>
      function) {
return new ConditionalAction<>(condition, function);
}
```

L7

```
private Optional<Function<S, T>>
   findMatchingFunction(S value) {
return conditionalActions.stream()
.filter(entry -> entry.condition().test(value))
.findFirst()
.map(ConditionalAction::action);
}
```

{ IMPLEMENTING ORELSE }

With *findMatchingFunction* in place, implementing *orElse* is straightforward (Listing 8). It returns the value of type *T* if no match is found. The value field, potentially nullable, is encapsulated in an Optional. As *findMatchingFunction* itself returns an Optional too, we use *flatMap* to unwrap the outer Optional, ensuring we obtain an `Optional<T>` instead of an `Optional<Optional<T>>.` In cases where no matching *Function* is found, we provide a default *Function* that returns the *defaultValue* without applying any logic (supplied to the *orElse* of the Optional). Finally, we apply the *Function* to the value wrapped by the Conditional.

L8

```
public T orElse(T defaultValue) {
 return Optional.ofNullable(value)
 .flatMap(this::findMatchingFunction)
 .orElse(obj -> defaultValue)
 .apply(value);
}
```

Now that these methods are implemented, we can proceed to confirm the success of our unit tests (Image 2).

With these tests successfully passing, we can now proceed to incorporate additional unit tests for methods that are yet to be implemented. While the details of these tests are beyond the scope of this article, you can explore the full implementation on my GitHub page [1].

L9

```
public T orElseGet(Supplier<? extends T>
    supplier) {
Objects.requireNonNull(supplier);

return Optional.ofNullable(value)
.flatMap(this::findMatchingFunction)
.orElseGet(() -> obj -> supplier.get())
.apply(value);
}
```

{ IMPLEMENTING ORELSEGET AND ORELSETHROW }

After completing the implementation of *orElse*, the next step is to proceed with implementing *orElseGet* (Listing 9). This method shares similarities with *orElse*, but with a crucial distinction—it accepts a *Supplier* responsible for providing a value when no matching condition is found. Additionally, it's important to ensure that the supplied Supplier is not null.

```
✓ no condition matches -> return default value
✓ a condition matches -> apply matching function
```

Image 2

Given that orElseGet is intended to be lazy evaluated, we utilize `Optional.orElseGet` within this method, as it also follows lazy evaluation. We pass a Supplier to `Optional.orElseGet`, which

L10
```
public <X extends Throwable> T orElseThrow(
 Supplier<? extends X> exceptionSupplier) throws
X {
 Objects.requireNonNull(exceptionSupplier);

 return Optional.ofNullable(value)
 .flatMap(this::findMatchingFunction)
 .orElseThrow(exceptionSupplier)
 .apply(value);
}
```

L11
```
public <U> ConditionalAction<S, U> and(Function<T,
    U> extraAction) {
return new ConditionalAction<>(condition,
action.andThen(extraAction));
}
```

L12
```
public <U> Conditional<S, U> map(Function<T, U>
   mapFunction) {
Objects.requireNonNull(mapFunction);

var updatedConditionalActions =
conditionalActions
.stream()
.map(condAction -> condAction.and(mapFunction))
.toList();

return new Conditional<>(value,
   updatedConditionalActions);
}
```

returns a *Function* responsible for extracting the value from the *Supplier* provided to the method. Subsequently, the *Function* obtained from the Optional pipeline is applied to the value wrapped by the Conditional.

The implementation of *orElseThrow* is similar to *orElseGet* but distinguishes itself by accepting a *Supplier* for providing a *Throwable* (Listing 10).

{ IMPLEMENTING MAP AND FLATMAP }

Map and *flatMap* apply transformations in the functional pipeline. The key difference is in their parameters: map takes a *Function* mapping an object from one type to another, while *flatMap* takes

L13
```
public <U> Conditional<T, U>
flatMap(Function<T, Conditional<T, U>>
      flatMapFunction) {
return map(flatMapFunction)
   .orElseGet(Conditional::empty);
}

private static <S, T> Conditional<S, T> empty() {
return new Conditional<>(null, Collections.
   emptyList());
}
```

a *Function* returning another Conditional. Similar to *Optional.flatMap*, the *flatMap* provided by the Conditional unwraps the outer monad when dealing with a *Function* returning another Conditional, preventing a nested monad within the functional pipeline.

To transform within our Conditional, we enhance each *ConditionalAction* in the conditionalActions field using Java's Function interface and its *andThen* method (which is used to combine Functions). We add a new method to *ConditionalAction* (Listing 11) that combines the existing condition and action with an additional action, mapping from T (the return type of the current action) to U (the return type of the extra action).
Now we can use this *and*-method in our *map* implementation (Listing 12).

To implement *flatMap*, we reuse the *map* method (Listing 13). Calling *map* with the *flatMapFunction* variable yields a `Conditional<S, Conditional<T,U>>`. To unwrap the outer Conditional, we chain this Conditional with *orElseGet*, which makes *flatMap* return a `Conditional<T, U>`. If no matching *ConditionalAction* is found, the method returns an empty Conditional using a convenience method (empty).

{ CONCLUSION }

I hope that this exploration has ignited curiosity within you, fellow developers, encouraging you to delve deeper into functional programming concepts and the fun journey of crafting custom monad-like types. The complete implementation of the Conditional, along with all the accompanying tests, can be found on my GitHub page [1]. Happy coding! ‹

{ REFERENCES }

https://github.com/LvdKooi/javamag2024

KUBERNATES

Implementeer je eigen Operator

Digipoort wordt herbouwd volgens het event-driven architectuurmodel, met Kubernetes als container platform. Digipoort kent een complexe infrastructuur met veel configuraties en een applicatie landschap dat dynamisch verandert. Om dit beheersbaar te maken is er een Kubernetes Operator geïmplementeerd, die het grootste deel van dit werk automatiseert. Het doel van dit artikel is om de opgedane kennis te delen en om te laten zien dat het implementeren van een eigen Operator eigenlijk heel toegankelijk is.

Rogier Pijpers is door Capgemini gedetacheerd bij Logius. Hier werkt hij als Lead Developer & Architect voor het herbouw traject Digipoort.

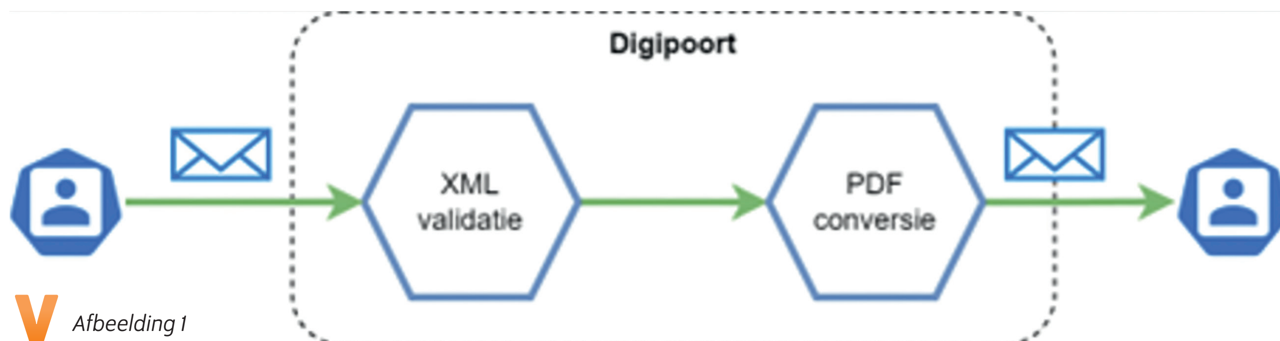
Georgina Fábián is een zelfstandig Solution Architect gespecialiseerd in cloud-based, event-driven systemen.

{ CASUS }

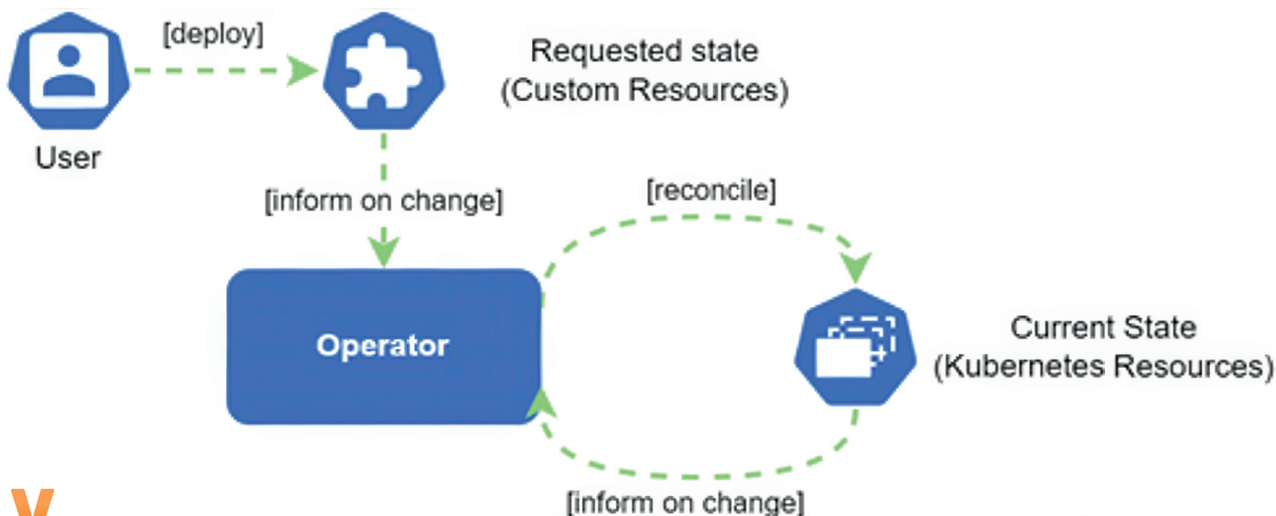
Aan de hand van de casus van *Digipoort*, een berichtenbus tussen bedrijven en overheidspartijen, laten we de globale werking van de Operator zien. Het bericht wordt door een reeks services verwerkt, afhankelijk van het berichttype. Zo'n verwerkingsservice heet in Digipoort een *IService* die als een microservice wordt gerealiseerd. Een bericht van een bepaald type wordt bijvoorbeeld behandeld door een *XML validatie service*, gevolgd door een *PDF conversie service*, zie afbeelding 1.

Afhankelijk van het type bericht worden er andere reeks IServices gebruikt. De reeks IServices heet in Digipoort een *IProces* die de verwerkingstroom van een specifieke berichttype bepaalt. Binnen een IProces worden de IServices aan elkaar gekoppeld door middel van *Kafka topics*.

De IProcessen zijn configuraties die in *deployment time* bekend zijn. De IProces specificaties bevatten hun *SLA*'s, die bepalen of deze met gedeelde IService instanties werken of eigen dedicated IService instanties vereisen. De berichtenflows ontstaan tijdens deployment, door de juiste IService instanties met de juiste Kafka topics te koppelen. *Run-time orchestratie* is niet nodig, omdat de topologie van de service's en de Kafka topics samen voor de juiste verwerking van de berichten zorgen.



Afbeelding 1



Afbeelding 2

{ DE KUBERNETES OPERATOR }

Het *Kubernetes* platform is in grote mate uit te breiden. Dit kan op verschillende manieren, maar een veelgebruikte manier is door middel van *Custom Resource Definitions* (CRDs). Dit is een uitbreiding op de Kubernetes API, die je in staat stelt om eigen Kubernetes resources te definiëren. Deze API uitbreidingen kunnen vervolgens gecombineerd worden met automatisering in de vorm van een Operator. Een "Kubernetes Operator" is eigenlijk een design pattern, bedoeld om handmatig werk van een beheerder (operator) te automatiseren.

Het pattern werkt grofweg als volgt. Als input neemt de Operator de *requested state* (de staat waarin het applicatie landschap dient te verkeren, omgeschreven door de Custom Resources) en de *current state* (de huidige staat van Kubernetes Resources). Vervolgens zorgt de Operator ervoor dat de current state steeds bijgewerkt wordt naar de requested state, door middel van wijzigingen die de control loops uitvoeren.

Er zijn een aantal hulpmiddelen beschikbaar voor het ontwikkelen van Kubernetes Operators, zoals de *Operator SDK*. Met goed begrip van het *operator pattern* kan een Operator in elke programmeertaal ontwikkeld worden. Gezien het ecosysteem van Digipoort is deze operator specifiek gebouwd in Java 17 in combinatie met Spring Boot en de Fabric8 Kubernetes Client [1].

{ REQUESTED DECLARATIVE STATE }

De *requested declarative state* is een omschrijving van de gewenste werkelijkheid door middel van CRDs. Voor Digipoort gebruiken wij IProces en IService CRDs, die door de Operator omgezet worden van omschrijving naar werkelijkheid.

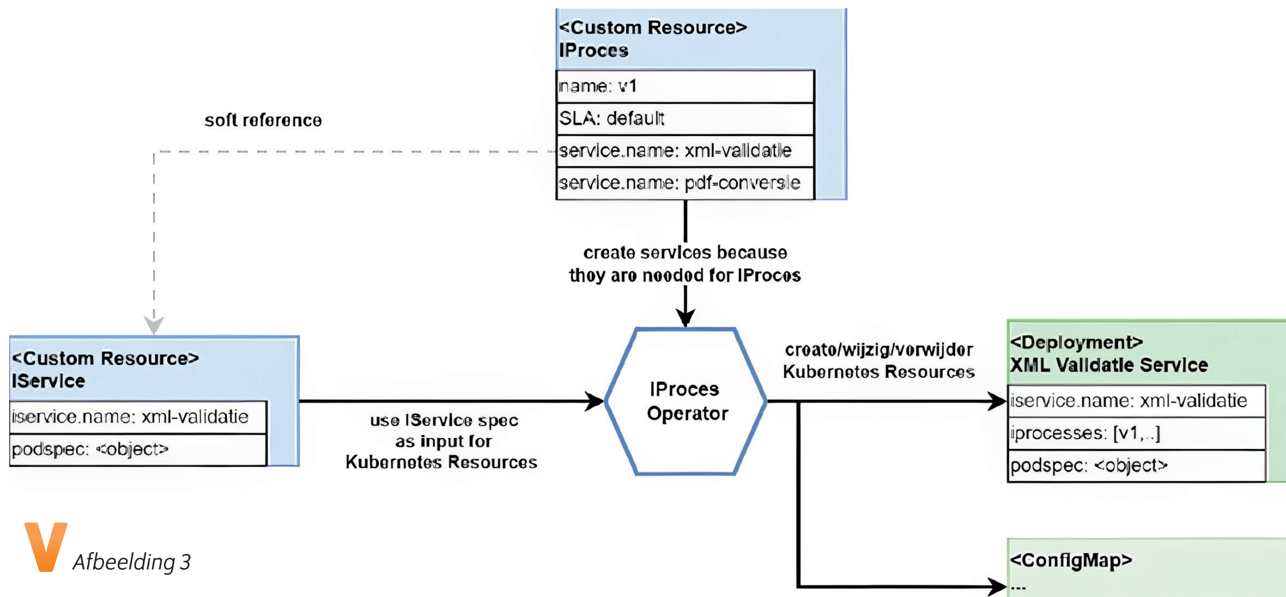
De IProces resource bevat een lijst van IService namen, samen met alle metadata die nodig is om het verwerkingsproces te realiseren. De IService namen verwijzen naar de namen van de IService resources. De IService resource bevat een pod-specificatie van een verwerkingsservice, op basis waarvan een Kubernetes *Deployment* gemaakt kan worden. De Operator gebruikt de Custom Resources als input: wanneer er een nieuw IProces Resource wordt toegevoegd, dan creëert of wijzigt de Operator voor elke benodigde IService de bijbehorende Kubernetes Deployment en andere resources.

Wanneer een IService definitie in geen enkel IProces gebruikt wordt, dan wordt er dus ook geen Deployment voor die IService aangemaakt. Op die manier worden IServices alleen in het leven geroepen als ze daadwerkelijk in een IProces gebruikt worden. De Operator verwijdert ook bestaande IService Deployments als er geen IProcessen meer zijn die gebruik van de betreffende IService maken. Alle Kubernetes Resources die de Operator heeft aangemaakt houdt hij in de gaten door middel van een label. Als een wijziging plaatsvindt op een van deze resources dan worden de relevante control loops uitgevoerd, om ervoor te zorgen dat de wijziging in lijn is met de requested state, zie afbeelding 3.

{ VOORBEELD VAN IPROCES CRD NAAR CUSTOM RESOURCE }

Met de CRD wordt een schema gedefinieerd als uitbreiding op de Kubernetes API. Aan de hand van dit schema kunnen custom Kubernetes objecten gemaakt worden. De CRD kan op Kubernetes toegepast worden met `kubectl apply -f <filename>`. De link naar de Kubernetes documentatie waarin staat hoe je een CRD opbouwt kun je vinden in de referenties. Nu het CRD schema bekend is in Kubernetes kunnen we een custom resource maken van type IProces met de naam *iproces-1*, Listing 1.

De Operator gebruikt deze resource als requested state en maakt aan de hand hiervan de benodigde objecten aan. De IProces Resource bevat in dit voorbeeld twee IServices. Op basis hiervan worden *Deployments*, *ConfigMaps* en *HorizontalPodAutoscalers* aangemaakt (en andere objecten zoals Kafka topics en S3 buckets).



Afbeelding 3

L1

```
apiVersion: "digipoort.logius.nl/v1"
kind: IProces
metadata:
  name: iproces-1
spec:
  name: iproces-1
  services:
    - name: "xml-validatie" # uses default
      replicas min:1 and max:1
    - name: "pdf-conversie"
      minReplicas: 1
      maxReplicas: 5
```

{ CONTROL LOOPS }

Door middel van control loops zorgt de operator ervoor dat de werkelijkheid overeenkomt met de requested state. Hiervoor bestaan twee soorten control loops.

Requested state control loop: Deze worden getriggered op het moment dat een wijziging wordt gemaakt in de IProces of IService custom resources. Dit betekent dat de verwachte werkelijkheid is veranderd.

Current state control loop: Deze worden getriggered wanneer resources die beheerd worden door de operator zijn gewijzigd. Bijvoorbeeld wanneer een Deployment verwijderd wordt door een externe factor.

De triggers van deze control loops worden aan de Operator doorgegeven via *Informers*. Een Informer geeft een signaal als een bepaald type resource gewijzigd wordt. Het is hier aan de developer om te bepalen welke logica op basis van dit signaal uitgevoerd wordt.

De uit te voeren logica is verdeeld en toegekend aan verschillende reconcilers. Je zou alle reconcilers tegelijk uit kunnen voeren bij elk signaal, dit is makkelijk en te begrijpen, maar niet erg efficiënt: de meeste control loops zijn waarschijnlijk niet relevant voor de gewijzigde resource type. Niet alleen genereert elke call naar Kubernetes load op de Kubernetes API, het zal de performance van de Operator ook niet ten goede komen.

Door efficiënt met de code in de EventHandlers om te gaan, kun je dus alleen reconcilers uitvoeren die relevant zijn voor de gewijzigde resource. Bijvoorbeeld: een IService heeft een Deployment en een ConfigMap. Op het moment dat het Deployment object wordt gewijzigd, hoeft alléén de control loop uitgevoerd te worden die controleert of alle verwachte Deployments nog bestaan. Omdat er geen ConfigMaps gewijzigd zijn weten we dat deze nog de correcte staat hebben.

Het template in Listing2 geeft weer hoe een event handler alleen getriggered wordt wanneer een Deployment is gewijzigd.

Door gebruik te maken van de resources, supertypes en interfaces in de Fabric8 Client kun je dit zo fijnmazig maken als noodzakelijk. HasMetadata is een interface dat geïmplementeerd wordt door elke Kubernetes resource die metadata bevat. HasMetadata heeft een supertype KubernetesResource die door elke Kubernetes resource geïmplementeerd wordt.

Een Deployment bevat metadata en implementeert dus het interface HasMetadata. Ik kan mijn eventhandler op deze manier zo specifiek maken als ik wil. De `ResourceEventHandler<Deployment>` in bovenstaand voorbeeld is alleen toepasbaar op het Deployment type. Als ik een

L2

```
// informer configuration
client.apps().deployments()
  .inNamespace("my-namespace")
  .inform(new ResourceEventHandler<Deployment>() {
      ...
      @Override
      public void onUpdate(HasMetadata resource,
HasMetadata otherResource) {
       // run my reconciliation loop only for
       Deployments, not for ConfigMaps
      }
      ...
   });
```

L3

```
var eventHandler = new InformerEventHandler
   (reconcilers); // pass list of relevant
      reconcilers here

client.configMaps()
  .inNamespace("my-namespace")
  .inform(eventHandler);
```

`ResourceEventHandler<HasMetadata>` implementeer, dan kan ik dezelfde eventhandler toepassen op elke resource die metadata bevat, enzovoort. Binnen Digipoort maken we hier gebruik van door één generieke eventhandler te gebruiken. Alle resources waar wij gebruik van maken hebben metadata, dus implementeren we de eventhandler op het niveau van HasMetadata. Per eventhandler configureren we de relevante reconcilers, Listing 3.

{ RECONCILERS }

De Eventhandlers voeren *Reconcilers* uit op de beheerde objecten, om ervoor te zorgen dat de current state correct is en overeen komt met de requested state. Een reconciler bestaat over het algemeen uit drie verschillende controles, Listing 4.

Missing resources: Er worden bepaalde resources verwacht die niet bestaan in de current state. Voor deze resources wordt een create uitgevoerd.

Unmanaged resources: Er bestaan resources met de tag *managed-by=my-operator* die niet verwacht worden. Deze resources worden verwijderd.

Invalid resources: Resources die bestaansrecht hebben, maar inhoudelijk niet overeen komen met deverwachting. Hierop wordt een update/replace uitgevoerd met de verwachte staat.

{ TOT SLOT }

In een gedistribueerde microservice wereld raakt configuratie snel gecompliceerd en verspreidt over het landschap. Als gevolg hiervan wordt de omgeving moeilijker te beheren, waarmee de kans op fouten oploopt. Kubernetes helpt je om de beheer taken te automatiseren, door de uitbreidbare opzet van het platform. Door gebruik te maken van API extensions en het Operator pattern wordt het beheer makkelijker, inzichtelijker en minder foutgevoelig. De declaratieve requested state door middel van CRD's zorgt voor een gemakkelijk begrijpbare omschrijving van het systeem. In het geval van Digipoort hoeven de DevOps engineers uitsluitend nog de requested state te beheren, terwijl het platform zorgt voor de deployments en de configuratie. ‹

L4

```
// voorbeeld: create if missing

expectedResources.stream()
   .filter(resource -> !actualResources.contains
      (resource))
   .forEach(resource ->
      client.apps().deployments()
         .inNamespace("my-namespace")
         .resource(resource)
         .create()
      );
```

{ REFERENTIES }

https://github.com/fabric8io/kubernetes-client
https://kubernetes.io/docs/concepts/extend-kubernetes/#extension-patterns
https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
https://sdk.operatorframework.io/
https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/

NLJUG INNOVATION AWARD 2024

JAVA DEVELOPERS MAKEN HET VERSCHIL

IN INNOVATIEVE TECHNOLOGISCHE ONTWIKKELINGEN

Is jouw bedrijf een innovator en heb jij een succesvol, innovatief, verantwoord en/of omvangrijk project? Dan is dit jouw kans om in de schijnwerpers te staan! De NLJUG Innovation Award wordt voor de zesde keer uitgereikt tijdens J-Fall en we zijn op zoek naar het meest innovatieve project. Elke NLJUG businesspartner dat een project inschiet, krijgt tevens 2 J-Fall conferentie-tickets! Maak kans op een timeslot binnen het programma van J-Fall en inschrijving door je project voor maandag 16 september 2024 in te dienen!

Winnaars

- **2018** Yolt
- **2019** Tennet
- **2021** Picnic
- **2022** Omoda
- **2023** Triopsys

INSCHRIJVING

Vanaf nu tot en met 16 september 2024 kan een project en/of persoon worden voorgedragen. Ken jij een bedrijf, project en/of persoon die dit jaar de NLJUG Innovation Award zou moeten winnen?

Schrijf je in op nljug.org/nljug-innovation-award/

BYTE SIZE

Website Carbon Calculator
How does it work? FAQ Get the badge! API Consultancy
Website carbon results for: nljug.org
D
Oh no! This web page achieves a carbon rating of D
This is dirtier than 53% of all web pages globally
Global average
A+ A B C D E F
Learn about our rating system
This page was last tested on 27 Mar, 2024. Test again
Copy URL
Oh my, 0.51g of CO2 is produced every time someone visits this web page.
How do we calculate this?

{ THE ORIGINAL WEBSITE CARBON CALCULATOR }

Website Carbon allows you to estimate your web page carbon footprint. It generates a report with tips and tricks to reduce the footprint (by switching to smaller images, more efficient file formats or deleting unused JS/CSS for example). It also offers a badge so you can show the (low) footprint of your own site - see the footer of **https://hanno.codes** for an example of this. **https://www.websitecarbon.com/**

> **Hanno Embregts**

{ SPRING TIPS: SPRING BOOT TESTJARS }

During an internal hackaton session we had a look at Spring Boot testjars. A (partial) alternative to testcontainers. It can help simplify integration testing in a microservices architecture with multiple spring boot bases services. **https://spring.io/blog/2024/02/08/spring-tips-spring-boot-testjars**

> **Thomas Zeeman**

{ MODERN WEB BLOAT MEANS SOME PAGES LOAD 21MB OF DATA }

Earlier this month, **Danluu.com** released an exhaustive 23-page analysis/op-ed/manifesto on the current status of unoptimized web pages and web app performance. For example, the Wix webpage requires loading 21MB of data for one page, while the more famous websites Patreon and Threads load 13MB of data for one page. This can result in slow load times that reach up to 33 seconds! **https://tech.slashdot.org/story/24/03/19/218216/modern-web-bloat-means-some-pages-load-21mb-of-data**

HOW TO WRITE A BYTE SIZE' BYTE SIZE

Do you have a tip to share, an opinion, or simply want to share something short? That's what the byte size format is for! Byte sizes are short bits of information, in a Twitter format: you have roughly 280 characters, and space enough for a link, or a picture! Send us your byte sizes by @JavaMagazineNL on Twitter or by mailing **info@nljug.org**!

> **Java Magazine Editorial Committee**

Vooruitblik J-Spring 2024

CHILIT is dit jaar weer van de partij als hoofdsponsor bij J-Spring en we hebben uiteraard een vol programma met activiteiten voor jullie in petto!

HOT SAUCE CHALLENGE

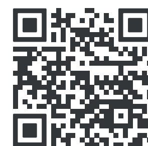
Voor de volgende editie van de Hot Sauce Challenge zullen we wederom een vurige reeks gesprekken ontsteken met collega developers. Naarmate elke ronde saus heter wordt, zal duidelijk worden wie het grootste heethoofd is. We zijn ook verheugd om onze exclusieve Signature Hot Sauce opnieuw te presenteren, die vorige keer al favoriet was en gegarandeerd weer een unieke draai aan de gesprekken zal geven. Doe vooral mee voor de prijs, of omdat je het leuk vindt om gewoon even met ons te spreken. Het mag uiteraard ook beide zijn.

AI MINIGAME

ONTWIKKEL JE EIGEN GAME AI!

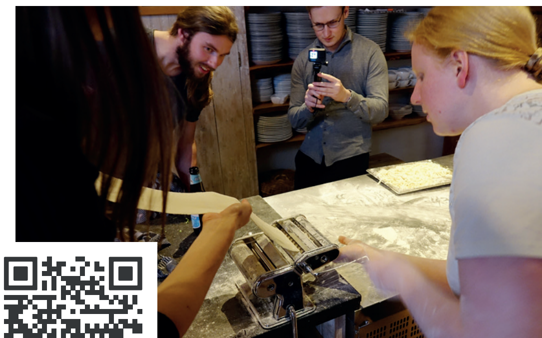
Bij onze stand zullen bezoekers de uitdaging aangaan om hun eigen Game AI te ontwikkelen. Deze activiteit biedt een hands-on ervaring van game development, in een vorm waar deelnemers hier laagdrempelig mee aan de slag kunnen gaan. De creativiteit en innovatie van de deelnemers zullen centraal staan. De competitie zal stevig, maar vriendschappelijk worden en we zijn erg benieuwd naar de creatiefste oplossing! Dit is een ideale gelegenheid om jouw vaardigheden te verfijnen en eens te proeven hoe het is om een stukje van je eigen game te maken.

Uiteraard valt er ook wat te winnen: Degene die de meest effectieve AI schrijft, wint een LEGO Pacman set!



LEKKERNIJEN

Bij onze stand hebben we ook dit jaar weer iets tegen de afterlunchdip: nog meer eten! Wat het precies wordt, houden we nog even geheim, maar reken maar dat het lekker wordt. Perfect om even bij te komen van alle activiteiten. Kom dus zeker langs om te proeven wat we voor je in petto hebben!



EVEN VOORSTELLEN:

Wij zijn **CHILIT**, een groep Java developers met hart voor het vak. Wat we doen, doen we goed. We begrijpen elkaar, en begrijpen ook dat het belangrijk is om met nieuwe, moderne technieken te werken. Denk aan: cloud-native libraries/frameworks en de nieuwste versies van Java, Spring Boot en Quarkus. Met twee Java developers aan het roer, weten we wat leuke en juist minder leuke projecten zijn. Dus we zorgen dat we bij onze klanten ook dit soort projecten uitzoeken.

Lijkt het je leuk om eens kennis met ons te maken - onder het genot van een kopje koffie of digitaal - dan lijkt ons dat hartstikke leuk! Spreek ons aan bij de stand, bel ons, mail ons of stuur een postduif (voor instructies zie RFC 1149), dan plannen we wat in.

WERKEN BIJ CHILIT?

Chilit

KEYNOTE:

From Problem to Performance: A Case Study on Java Performance

In the world of software development, selecting the right tools and frameworks is only part of the story; a significant amount of our time is devoted to solving complex bugs and performance issues. These problems are often difficult to debug and are caused by underlying systems that we typically take for granted. This keynote will explore several common scenarios that lead to unexpected performance issues, through a detailed case study that I have experienced. We will handle a specific example of a mortgage batch process that got stuck, and walk through the diagnosis step by step. The focus will be on the importance of thorough preparation for the production phase and the need to proactively identify and address performance issues. Furthermore, we will discuss solutions and strategies to detect and prevent these problems in the future. By the end of this session, you will have gained practical knowledge on recognizing and resolving performance bottlenecks, and understand the importance of being proactive in your approach to ensure the stability and efficiency of Java applications.

TIMESLOT:

Mastering Java Performance: From Monitoring to Profiling

This session builds on the principles discussed in the keynote and delves deeper into the hierarchy of performance bottlenecks in Java applications. We will address more complex examples of performance degradation in various systems such as frameworks, databases, and concurrency management. The focus is on the use of advanced diagnostic tools and profiling techniques such as VisualVM, Java Flight Recorder, and tools within IntelliJ. These tools can assist you in quickly and effectively detecting and resolving performance issues. Additionally, we will discuss practical examples that illustrate how performance issues manifest (typically only in production), such as exhausted connection pools due to API calls within a transaction and full table scans caused by inadequate indexing. We conclude the session with a clear action plan for setting up effective monitoring and making potential bottlenecks visible. After this presentation, you will have more tools under your belt to anticipate and quickly resolve performance issues.

June 13th, 2024
Jaarbeurs Utrecht

DEVITJOBS.NL

Java Ontwikkelaar
Salarisindicatie:
35.000 - 65.000 EUR

Locatie:
Weesp

Technologieën:
Java, CI/CD, JavaScript

QA Engineer Medior (Nederlandstalig)
Salarisindicatie:
45.000 - 65.000 EUR

Locatie:
Den Haag

Technologieën:
Cypress, ISTQB, JavaScript, Protractor, Selenium, TMap, Tosca, TypeScript

Java Developer Integrations Tribe
Salarisindicatie:
50.000 - 80.000 EUR

Locatie:
Groenlo

Technologieën:
Spring, Datadog, HL7, Java

Senior Java developer
Salarisindicatie:
110.000 - 140.000 EUR

Locatie:
Zeist

Technologieën:
Spring Boot, Spring, Kubernetes, SQL, PostgreSQL, CI/CD, Backend, Cloud, Docker, DevOps, Git, GitHub, GitLab, Hibernate, Helm

Senior Java Developer
Salarisindicatie:
50.000 - 80.000 EUR

Locatie:
Nijmegen

Technologieën:
Java, Docker, Kubernetes, Java EE, Spring

Java Developer
Salarisindicatie:
35.000 - 65.000 EUR

Locatie:
Rijswijk

Technologieën:
Java, Spring Boot, Angular, Vaadin, React, MongoDB, MySQL, IntelliJ

Find all these vacancies and more

at Devitjobs.nl

RITD

The Future of AI: Consciousness, Djano, and the Evolution of Technology

SPEAKER: ANDRIJANO DJENISOV

In this engaging lecture, we delve into the fascinating world of artificial intelligence (AI), consciousness, and the role of Djano as a bespoke AI solution. We begin by exploring the definition of AI and how it has evolved from an abstract concept to a powerful tool transforming our society.

Next, we delve into the concept of consciousness and the role it plays in the evolution of AI. We examine the question of whether AI can ever achieve consciousness and how this may change our understanding of intelligence and self-awareness. Finally, we examine Djano, a leading AI platform offering tailored solutions for businesses in various sectors. We discuss how Djano pushes the boundaries of AI by combining advanced algorithms and deep analysis with empathy and understanding of human experiences.

Whether you're a technology enthusiast, a business leader seeking innovative solutions, or simply curious about the future of AI, this lecture provides an engaging and inspiring insight into the power of technology and the human mind.

POSTKAMER
Wed 12:00 - 12:45

Kvk

Getting smart with KVK's synthetic data

SPEAKER: WASEEM SADIQ

Join us for an enlightening session where Waseem Sadiq delves into the innovative use of synthetic data at the KvK (Chamber of Commerce). This presentation will explore the fascinating intersection of data privacy and open government data—fields often seen at odds. Discover how synthetic data serves as a revolutionary solution, providing privacy-compliant data that retains the authenticity necessary for robust data analysis and integration. Learn about the KvK's current projects and future plans to make this synthetic data publicly available, enabling the market to leverage it for diverse use cases. Don't miss this opportunity to see how synthetic data is shaping the future of data privacy and open data initiatives!

POSTKAMER
WED 14:30 - 15:15

Nationale Nederlanden

NN's Successful Application of ChatGPT Technology for Automated Call Logging

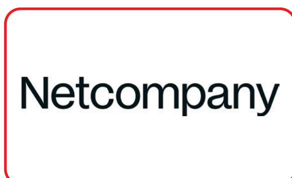
SPEAKERS: NIKKI DOORHOF & ROMERA HILLEBRANDT

Two years before ChatGPT was launched, NN Group tested OpenAI's model for the Dutch language. The experiment was successful, and OpenAI asked NN Group to test other AI models, including ChatGPT. The team spent 1.5 years experimenting with ChatGPT to ensure the safe use of the technology without giving away company data. Now, NN Group is one of the first companies in Europe to apply ChatGPT technology on a large scale in production for Automated Call Logging. Learn about how they applied ChatGPT, got the business on board, and the challenges they faced. Get inspired to learn more about the potential of AI in business.

JUTEKELDER
Wed 11:00 - 11:45

Netcompany

Green Software: If we change the way we code, then the Climate doesn't have to change



SPEAKER: EDWIN VAN DEN HOUDT

Enterprise technology by itself is responsible for emitting approximately 1% of global emissions. Let's put this into perspective. This amount matches half of the emissions generated by aviation or shipping industries. Even more remarkably, it's similar to the entire carbon footprint of the United Kingdom.

So, what can we do about this? This talk will share practical green software development principles that you can implement in your project to minimize the environmental impact of your solution. Green software is an emerging discipline at the intersection of climate science, software design, electricity markets, hardware, and data center design.

At its core, green software is all about carbon efficiency. It's designed to emit the smallest carbon footprint possible. But how is this achieved? It's a careful blend of three key strategies: energy efficiency, carbon awareness, and hardware efficiency. We'll delve into these essential concepts, giving you the tools to incorporate them into your own processes. Join us, as we navigate this crucial and exciting journey towards sustainable software practices.

MEELZOLDER
WED 14:30 - 15:15

Rabobank

How to write maintainable code in a large organization

SPEAKER: RAGNA GERRETSEN

What if you're working in a large organization, doing the same thing but separately? How can you work together to become an efficient organization? Within Rabobank we have many departments who try to solve this problem, from re-usable pipeline templates to IT4T teams. But what about reusing code? We all want to be autonomous and be masters of our own code however this comes at a cost: code duplication. In this presentation I will start with sharing some of the low hanging fruit to remove duplication and will end with how to efficiently work together to reduce duplication. I will demonstrate the use of these concepts in a Java-based Spring Boot application in a microservices architecture and provide some examples of best practices and potential pitfalls. By the end of this presentation, you will have a better understanding of how to leverage each of these concepts to simplify and improve your development process across your organization. Most importantly you will learn how to reduce your code and find the right abstraction, in turn creating happy developers.

DOUCHELOKAAL
Wed 12:00 - 12:45

ABN AMRO

Tulips to Turmeric: Lessons Learned from a Global Team

SPEAKER: RIJO SAM

Ever wondered why working in a global team feels like unravelling a mystery? You're not alone! The reality of working within such a team can differ from expectations. Often, the greatest challenge for developers isn't the actual work, but rather navigating the distance, understanding cultural disparities, and maintaining meaningful connections with team members. From unexpected challenges to surprising growth opportunities, this talk aims to explore the secrets behind successful distributed teamwork. Throughout the session we'll explore practical strategies to enhance communication, bridge cultural gaps, and foster strong collaboration across continents. Join me as we embark on a journey filled with laughter, learning, and limitless possibilities.

DOUCHELOKAAL
WED 14:30 - 15:15

3 REDENEN OM ALS DEVELOPER BIJ DE RIJKSOVERHEID TE WERKEN

Als developer bij de Rijksoverheid raakt jouw werk het leven van bijna 18 miljoen Nederlanders. Van de vooraf ingevulde aangifte (VIA) tot het aanvragen van studiefinanciering. Van het betalen van invoerrechten tot het vinden van smokkelwaar op containerschepen. Niet alleen de inhoud van het werk maakt een baan als IT'er bij een van de rijksorganisaties interessant. Ook de goede arbeidsvoorwaarden en de fijne werksfeer maken dat de Rijksoverheid veel te bieden heeft als werkgever voor digitaal talent.

1. STEEDS MEER VROUWELIJKE COLLEGA'S

Lisa Noorlander, ontwikkelaar Belastingdienst: 'De Belastingdienst geeft mij vrijheid en voldoende mogelijkheden om me intern te ontwikkelen. We hebben veel expertise in huis. Het is een mannenwereld, maar dat is voor mij inmiddels vanzelfsprekend en doet niets af aan prettig samenwerken. In mijn vorige team was ik de enige vrouw, inmiddels heb ik enkele vrouwelijke collega's. Het zou leuk zijn als steeds meer vrouwen voor de IT kiezen, het is een prachtig vak.'

2. GOEDE ARBEIDSVOORWAARDEN

Miel van Welzen, Java-developer Kamer van Koophandel (KVK): 'Bij de Rijksoverheid lever je direct een bijdrage aan de vitale infrastructuur van Nederland. Zonder de dienstverlening van de KVK kan het bedrijfsleven in Nederland niet draaien. Bedrijven en overheidsinstanties, maar ook bijvoorbeeld de rechtspraak zijn afhankelijk van onze data. Het voelt goed om je kennis in te zetten voor een maatschappelijk doel. Daarnaast zijn de arbeidsvoorwaarden goed. We hebben als zelfsturend team veel vrijheid, vast gereserveerde tijd voor innovatie en ruim voldoende vakantiedagen.'

3. STERKE BAND BINNEN TEAM

Melanie, software developer bij de Algemene Inlichtingen en Veiligheidsdienst (AIVD): 'Op feestjes zijn mensen meestal toch niet zo geïnteresseerd in alle details van je IT-baan, tenzij ze zelf ook in IT werken. Als iemand me toch specifieke vragen stelt, heb ik een coverstory. Ik kan wel praten over technische dingen — welke tools en programmeertalen ik gebruik bijvoorbeeld — maar noem een andere werkgever. Er heerst een gevoel van: we moeten dit samen doen. Juist omdat we niet over ons beroep kunnen praten met de buitenwereld, is er automatisch een sterkere band binnen het team.'

STAND EN TALKS RIJKSOVERHEID

Maak tijdens TEQNation op 22 mei kennis met IT'ers bij verschillende rijksorganisaties in de stand van de Rijksoverheid. Dit jaar vind je daar developers van de Belastingdienst, de Nederlandse Voedsel- en Warenautoriteit (NVWA), Logius, de Kamer van Koophandel (KVK), het Rijksinstituut voor Volksgezondheid en Milieu (RIVM) en de Algemene Inlichtingen- en Veiligheidsdienst (AIVD).

Waseem Sadiq van de KVK verzorgt een presentatie met de titel: Getting smart with KVK'synthetic data in de zaal de Postkamer tussen 14:30 en 15:15 uur. Vergeet niet mee te doen in onze stand met de Kop van Jut en win een mooie prijs als je de hoogste score hebt.

CHECK ONZE IT-VACATURES

Lisa, Miel en Melanie vertellen meer over hun werk op **Werkenvoornederland.nl/ict**. Scan de QR-code voor de meest actuele IT-vacatures bij de Rijksoverheid.

Rijksoverheid

JAVA 22

Java SE 22 was released on March 19, 2024, so it should be generally available when this article is published. It boasts a set of 12 JEPs, finalizing features like 'unnamed variables and patterns' and previewing new ones like 'statements before super(...)' and 'stream gatherers'. Let's get into these features!

Ivo Woltring is a Principal Expert and Codesmith at Ordina and likes to have fun with new developments in the software world.

Hanno Embregts is an IT Consultant at Info Support with a passion for learning, teaching and making music. He has recently been named a Java Champion.

{ NEW FEATURES }

JEP 423: Region Pinning for G1 When working with JNI, functions like `GetStringCritical` and `ReleaseStringCritical` are utilized to acquire memory address pointers related to Java objects, which can then be managed accordingly. These pointers prevent the garbage collector from relocating the associated objects in memory to prevent pointer invalidation while they are active. However, in scenarios where the garbage collector lacks 'pinning' support, invoking any `Get*Critical` method temporarily halts garbage collection until all corresponding `Release*Critical` methods have been called. This temporary pause can have notable implications on memory usage and overall performance, depending on the application's behavior. This is why JEP 423 proposes the introduction of pinning functionality, thereby eliminating the necessity for garbage collection pauses starting from Java 22.

JEP 454: Foreign Function & Memory API Following extensive development within Project Panama and a series of many incubator and preview iterations, the Foreign Function & Memory API (FFM API) is now nearing completion under JEP 454. It enables Java to access external code, like functions in libraries written in different programming languages, and native memory (i.e. outside the JVM heap).

L1

```
#include <stdio.h>
int product(int a, int b) {
    return a * b;
}
int add(int a, int b) {
    return a + b;
}
int minus(int a, int b) {
    return a - b;
}
```

Designed to supersede the complex, error-prone, and sluggish Java Native Interface (JNI), the FFM API is expected to take away much of the implementation and performance problems with JNI (see Listing 1, 2).

JEP 456: Unnamed Variables & Patterns When matching a pattern or catching an exception where the involved variable isn't actually used, we now have the option to use an unnamed variable or pattern. It is indicated by the underscore character (see Listing 3).

JEP 458: Launch Multi-File Source-Code Programs Since Java 11 it has been possible to run Java classes with a `main` method directly with the java command without first having to compile them with `javac` as long as all code is contained within that single class file. This JEP extends the support by allowing to launch multi-file source-code programs (see Listing 4), allowing a simple starter project to postpone the introduction of build tools like Maven or Gradle to a later moment.

{ PREVIEW AND INCUBATOR FEATURES }

JEP 447: Statements before super(...) In Java 21 and earlier, a subclass constructor must always call the superclass constructor before doing anything else. But this restriction can sometimes get in the way (see Listing 5 for an example).

L2

```
import java.lang.foreign.Arena;
import java.lang.foreign.FunctionDescriptor;
import java.lang.foreign.Linker;
import java.lang.foreign.SymbolLookup;
import java.lang.foreign.ValueLayout;
import java.nio.file.Path;

public class JEP454 {
  public static void main(String[] args) {
  try (var arena = Arena.ofConfined()) {
   var lib = SymbolLookup.libraryLookup(Path.
of("calc.so"), arena);
   var linker = Linker.nativeLinker();
   var fd = FunctionDescriptor.of(ValueLayout.JAVA_INT,
       ValueLayout.JAVA_INT,
       ValueLayout.JAVA_INT);
   var addFunc = lib.find("add").get();
   var minusFunc = lib.find("minus").get();
   var multiplyFunc = lib.find("product").get();
   System.out.println("sum(20, 22)   = " +
     linker.downcallHandle(addFunc, fd).
invoke(20, 22));
   System.out.println("minus(45, 3)  = " +
     linker.downcallHandle(minusFunc, fd).
invoke(45, 3));
   System.out.println("product(7,6)  = " +
     linker.downcallHandle(multiplyFunc, fd).
invoke(7, 6));
  } catch (Throwable e) {
    throw new RuntimeException(e);
  }
  }
}
```

L3

```
// a few simple examples
// Example 1
try {
    int number = Integer.parseInt(args[0]);
} catch (NumberFormatException _) { // unnamed
variable
    System.out.println("Please enter a valid
number");
}
// Example 2
map.computeIfAbsent("key", _ -> new
ArrayList<>()).add("value"); // unnamed variable
// Example 3
switch (ball) {
    case RedBall _   -> process(ball);    //
Unnamed pattern variable
    case BlueBall _  -> process(ball);    //
Unnamed pattern variable
    case GreenBall _ -> stopProcessing(); //
Unnamed pattern variable
}
```

L4

```
//Greetings.java
public class Greetings {
    public String greet(String name) {
        return "Hello, " + name + "!";
    }
}
//JEP458.java
public class JEP458 {
    public static void main(String[] args) {
        System.out.println(new Greetings().
greet("World"));
    }
}
$ java JEP458.java
Hello, World!
```

L5

```
class StringQuartet extends Orchestra {
    public StringQuartet(List<Instrument>
instruments) {
        super(instruments); // Potentially
unnecessary work!
        if (instruments.size() != 4) {
            throw new
IllegalArgumentException("Not a quartet!");
        }
    }
}
```

L6

```
class StringQuartet extends Orchestra {
    public StringQuartet(List<Instrument>
instruments) {
        super(validate(instruments));
    }
    private static List<Instrument>
validate(List<Instrument> instruments) {
        if (instruments.size() != 4) {
            throw new
IllegalArgumentException("Not a quartet!");
        }
    }
}
```

L6

```
class StringQuartet extends Orchestra {
    public StringQuartet(List<Instrument>
instruments) {
        if (instruments.size() != 4) {
            throw new
IllegalArgumentException("Not a quartet!");
        }
        super(instruments);
    }
}
```

It would be better to let the constructor fail fast, by validating its arguments before the super(...) constructor is called. Pre-Java 22, we could only achieve this by introducing a static method that acts upon the value passed to the super constructor (Listing 6), but Listing 7 showcases the Java 22 way to achieve the same in a much more readable way.

JEP 457: Class-File API Java's ecosystem relies heavily on the ability to parse, generate and transform class files. Frameworks use on-the-fly bytecode transformation to add functionality transparently, typically bundling class-file libraries like ASM or Javassist to handle class-file processing. However, these libraries suffer because the six-month release cadence of the JDK causes the class-file format to evolve more quickly than before, meaning they might encounter class files that are newer than the class-file library they bundle.

To solve this problem, JEP 457 proposes a standard class-file API that can produce class files that will always be up-to-date with the running JDK. This API will evolve together with the class-file format, enabling frameworks to rely solely on this API, rather than on the willingness of third-party developers to update and test their class-file libraries.

The Class-File API, located in the *java.lang.classfile* package, consists of three main components:

Elements - Immutable descriptions of parts of a class file, such as instructions, attributes, fields, methods, or the entire file.
Builders - Corresponding builders for compound elements, offering specific building methods (e.g., ClassBuilder::withMethod) and serving as consumers of element types.
Transforms - Functions that take an element and a builder, determining if and how the element is transformed into other elements. This allows for flexible modification of class file elements.

JEP 457 [1] has a code example about the Class-File API.

JEP 459: String Templates String templates are now in second preview, to allow for more feedback from its users. The design of the feature prioritizes handling the possible dangers of string interpolation, as it is easy to construct strings for interpretation by other systems that potentially can be dangerously incorrect in those systems (think of SQL injection). Meet *template expressions*, a new kind of expression that offers string interpolation, but with safety in mind. Each template expression has to be processed by a specific template processor to ensure safe handling. Java offers three of them by default, the STR, RAW and FMT template processors.

Our article on Java 21 [2] has a code example about string templates.

JEP 460: Vector API In Java 22, the Vector API is submitted as an incubator feature for the 7th consecutive release. The Vector API will make it possible to perform mathematical vector operations efficiently. It's designed to enable accelerated computations on supported hardware without requiring any specific platform knowledge or code specialization. The API aims to expose low-level vector operations in a simple and easy-to-use programming model, allowing performance optimizations to be integrated seamlessly into existing Java code. A code sample can be found at [3].

JEP 461: Stream Gatherers The Stream API has become essential for Java developers, offering a powerful and concise programming paradigm. It consists of three main components: a *source of elements*, *intermediate operations*, and a *terminal operation*. While it provides a diverse set of predefined operations like mapping, filtering, and sorting, Java's language designers feel there's a need for more flexibility. This is why JEP 461 proposes *stream gatherers*.

Stream gatherers function similarly to the collect operation for terminal operations, but operate on intermediate streams. They introduce the gather operation that allows for various transfor-

mations of elements, such as one-to-one, one-to-many, and many-to-many mappings. This essentially makes every stream pipeline equivalent to the code in Listing 8.

The JEP also introduces several built-in gatherers, such as `fold`, `mapConcurrent`, `scan`, `windowFixed`, and `windowSliding`, which can be passed to the gather operation. It is also possible to implement a custom gatherer by implementing the `java.util.stream.Gatherer` interface. The JEP also suggests that future stream operations will be introduced as a built-in gatherer first to maintain the simplicity of the Stream API. Based on their proven usefulness, a built-in gatherer may be promoted to an intermediate operation in the future.

JEP 462: Structured Concurrency Structured concurrency is now in its second preview to allow for more feedback from its users. The feature enables better management of concurrent tasks in Java programs. With structured concurrency, tasks are organized into scopes to ensure that all tasks within a scope are complete before the scope itself is considered complete. This makes it easier to manage and control concurrent tasks, reducing the risk of problems such as race conditions and deadlocks. It also makes it easier to cancel or interrupt tasks within a scope without affecting other tasks, improving the overall stability and reliability of the program. In simple terms, structured concurrency helps developers write more reliable and efficient code when dealing with multiple tasks that run concurrently.

Ivo's article about Java 19 [4] has a code example about structured concurrency.

JEP 463: Implicitly Declared Classes and Instance Main Methods Implicitly declared classes and instance main methods are now in the second preview to allow for more feedback from its users. They allow for a much shorter implementation of the classic "Hello, World!" program (Listing 9), thereby preventing newcomers to Java from having to grasp concepts that they don't need on their first day of Java programming, like the public access modifier, the static modifier or the `String[] args` parameter.

JEP 464: Scoped Values Scoped values are now in the second preview to allow for more feedback from its users. They offer a way to share immutable data within and across threads, particularly beneficial when dealing with large numbers of virtual threads. Unlike thread-local variables, which have been available since Java 1.2, scoped values are immutable and exist only for a bounded period during thread execution. Thread-local variables, while useful for achieving thread safety in the past, come with drawbacks such as mutability, potential memory leaks, and inheritance by child threads, which could lead to significant memory usage when dealing with numerous virtual threads.

L8

```
stream
 .gather(...)
 .gather(...)
 .gather(...)
 .collect(...);
```

L9

```
void main() { // this is an instance main method
inside of an implicitly declared class
    System.out.println("Hello, World!");
```

Scoped values provide a solution to these issues and are recommended for scenarios where thread-local variables are currently used for transmitting unchanging data. Our article on Java 21 [2] includes a code example demonstrating the usage of scoped values.

{ CONCLUSION }

Apart from the 12 JEPs that we discussed, many other updates were included in this release, including various performance and stability updates. Our favorite language is clearly more alive than ever, and on top of that, it'll probably attract more newcomers due to the features that focus on starting projects. Here's to many happy hours of development with Java 22! ‹

{ REFERENCES }

1 https://openjdk.org/jeps/457
2 https://nljug.org/java-magazine/java-magazine-4-2023-20-jaar-j-fall-beursspecial/
3 https://github.com/IvoNet/java-features-by-version
4 https://nljug.org/java-magazine/2022-editie-4/java-magazine-4-2022-j-fall-is-here/

AUTOMATING LARGE-SCALE REFACTORINGS WITH ERROR PRONE

When it comes to writing good computer programs, avoiding mistakes is key. To prevent errors, several processes are generally put in place, such as thorough review procedures and the use of static analysis tools. However, despite these measures, human error remains a stubborn obstacle in software development. Issues frequently slip through the review process, and tools like Checkstyle, SonarCloud, and SpotBugs only highlight some common problems. This leaves the developer with the task of manually applying the suggested improvements.

Rick Ossendrijver is a Software Engineer at Picnic and part of the Java Platform team.

Stephan Schroevers has a bachelor's degree in Computer Science and a master's degree in Logic He is currently the Tech Lead for Picnic's Consumer Domain.

Static analysis tools are beneficial as they help detect bugs before they enter our codebase. They save time for those reviewing pull requests, as these issues don't need to be pointed out during a review. Even better, it saves the frustration of solving or pointing out the same issues over and over again. It gets even more frustrating if developers are pointing out trivial things in pull request comments as they lead to repeated useless discussions.

Observing the boost that such tools can provide, we began to ask ourselves: How can we achieve a similar level of automation for bug patterns that we frequently encounter, which are not covered by existing tools? And, can we transition from tools that merely identify problems to tools that actually fix them?

Let's explore Error Prone!

{ ERROR PRONE }

Error Prone is a static analysis tool for Java that catches common Java mistakes and flags them as compile-time errors. Originally developed by Google and open-sourced in 2012, Error Prone integrates with the Java compiler. As a compiler plugin, it is not tied to a specific build tool, meaning it works seamlessly with Maven, Gradle, Bazel, and other build systems. It performs its error checking after the compiler's flow analysis phase, ensuring that type checking is complete and all symbol information is available. By this stage, all errors detected by the standard compiler have been reported.

When Error Prone detects an issue, it is presented as a compiler warning or error. These diagnostics include a helpful message, a link to documentation, and suggested code to fix the issue. Additionally, Error Prone can generate patch files containing suggested fixes, which can optionally be applied directly to the code. This feature significantly simplifies the process of applying large-scale changes to a codebase.

{ EXAMPLES }

Consider a simple example where valid Java code doesn't quite achieve the desired outcome. It is easy to forget the 'throw' keyword in front of an exception initialization expression. The following code compiles, but no exception will be thrown when executed:

```
public static void main(String[] args) {
  if (args.length > 0 && args[0].isEmpty()) {
        new IllegalArgumentException();
  }
  // ... rest of the main code ...
}
```

When compiling this code with Error Prone configured, you'll encounter the following error:

```
> [ERROR] Example.java:[3,35][3,25]
[DeadException] Exception created but not thrown
> [ERROR] (see https://errorprone.info/bugpattern/
DeadException)
> [ERROR] Did you mean 'throw new
IllegalArgumentException();'?
```

The developer is not only presented a clear error message, but also provided a fully contextualised suggested fix. This is where Error Prone distinguishes itself as a static analysis tool.

Out of the box, Error Prone comes with an extensive set of over 500 checks, known as BugChecks. Examples include:
ArrayEquals: Flags comparing arrays by reference equality.
FormatStringAnnotation: Flags invalid format string usages.
Immutable: Flags type declarations annotated with @Immutable that appear mutable.
JUnitParameterMethodNotFound: Flags parameterized tests that reference a nonexistent provider method.
Developers can configure the checks as desired. For example, each check can be enabled or disabled, and warnings can be raised as errors or treated as suggestions.

In addition to the provided checks, Error Prone also allows you to create custom checks! This extensibility is what makes the tool even more attractive. As evidenced by the last example, checks can apply to any Java library. Error Prone built-in checks cover more than just JDK APIs, with checks for popular libraries like JUnit and Mockito. However, writing BugChecks means traversing and manipulating the Abstract Syntax Tree (AST) of the Java code, which isn't entirely trivial. Meaning, the developer needs to have some knowledge of Error Prone's APIs, and occasionally work with scarcely documented compiler APIs.

{ INTERNAL USE OF ERROR PRONE }

At Picnic, we use custom rules to catch common mistakes or incorrect usages of APIs. In figure Z, we flag an unnecessary application of `String.valueOf` to `"foo"`.

```
String str = String.valueOf("foo");
```

This results in the following compiler warning:

```
> [WARNING] Example.java:[18,31]
[IdentityConversion] This method invocation
appears redundant
> [WARNING] (see https://error-prone.picnic.tech/
bugpatterns/IdentityConversion)
> [WARNING] Did you mean 'String str = "foo";'?
```

As another example, for incorrect SLF4J log statements, we identify cases where the number of format string placeholders does not match the number of supplied arguments.

```
LOG.info("format-string-with-{}-{}-placeholders",
"value1");
```

This result in the following compiler warning:

```
> [WARNING] Example.java:[10,12]
[Slf4jLogStatement] Log statement contains
2 placeholder(s), but specifies 1 matching
argument(s).
> [WARNING] (see https://error-prone.picnic.tech/
bugpatterns/Slf4jLogStatement)
```

{ REFASTER}

Writing a check in Error Prone is not trivial. That's why Google created Refaster, which comes bundled with Error Prone. Refaster is a DSL that lets you define rewrite rules in Java code using before- and after-templates. First, these Java files are compiled into *.refaster* files. Then, the Error Prone compiler scans the codebase for code that matches the structure of a before-template. The matching code is then rewritten according to the associated after-template.

```java
static final class StringIsEmpty {
  @BeforeTemplate
  boolean equalsEmptyString(String str) {
    return str.equals("");
  }

  @BeforeTemplate
  boolean lengthEquals0(String str) {
    return str.length() == 0;
  }

  @AfterTemplate
  boolean after(String str) {
    return str.isEmpty();
  }
}
```

Above code is an example of a Refaster rule. A rule consists of at least one before-template and an after-template, indicated by the `@BeforeTemplate` and `@AfterTemplate` annotations. Note how in the example each template method has a parameter `String str`. Refaster treats such parameters as typed placeholders. When traversing the AST to find expressions that match the structure of any before-template, placeholders will match any subexpression of a compatible type. For example, the arbitrary piece of code `String.valueOf(getInteger()).length() == 0` will also be matched by the shown Refaster rule. When a match is found, Refaster automatically refactors the code as described in the after-template. After applying the shown Refaster rule to a code base, the `isEmpty()` method will be the only operator used to check whether a string is empty.

Google open-sourced the Refaster tool in 2014. While they haven't open-sourced any of their large internal body of Refaster rules, there is a GitHub issue created in 2017 [2], where they mention plans to open source (parts of) their Refaster rules. It's unlikely this will happen in the near future. Meanwhile, developers can still use Refaster's framework to create custom rules tailored to their codebases. At Picnic, we've created over 900 Refaster rules that are continuously applied to our codebase.

Refaster also supports advanced features. You can use generics to have Refaster rules match many (sub-)types. Conversely, you can define restrictions using predicates on what code should be matched. Refaster supports matching block statements in addition to individual expressions. Moreover, you can match arbitrary chunks of code by using a placeholder. The latter is especially useful for matching lambda expressions with a specific signature.

One downside of Refaster is that it can only patch expressions and statements, meaning it is limited to making changes in method bodies and (static) initializer blocks. This contrasts with the capabilities of Error Prone's BugChecks, which have full control over the AST and can perform larger refactoring operations.

{ CUSTOM BUGCHECKS }

BugChecks are more powerful than Refaster rules, but they're also more challenging to write.

```java
class A {
  public String method1() {
    return "";
  }

  public void method2() {
  }
}
```

The above listing shows a simple Java class and its abstract (and vastly simplified) representation in tree form, as seen in image 2, consisting of various nodes. Each node contains information about the code construct it represents. For instance, a method node specifies a method's name, return type, arguments, body, type arguments, and modifiers. Error Prone handles traversing the AST, allowing developers to define logic in the BugCheck to analyse, swap, edit, or remove a full node from the tree.

Imagine we want to remove empty methods from our codebase. Error Prone would analyse the classes and check each method tree node to see if it contains a body with statements. In the example, `method2` doesn't have a body, so that node can be dropped from the tree. This is a simple example, but Error Prone also supports more complex rewrites that make multiple changes throughout a file.

Error Prone performs analysis at the CompilationUnit level, meaning it analyses each source file individually. It can traverse the structure of the file and classes, like parent classes and interfaces, but it can't analyse multiple complete files at once. This means we cannot check whether `method2()` is actually used in other CompilationUnits.

{ EXTENSIBILITY OF ERROR PRONE }

As previously discussed, we can define custom BugChecks with Error Prone. When they are available on the annotation processor path, the plugin mechanism will dynamically load the plugins using Java's service loader mechanism.

There are some organisations like Mockito, Palantir, Chromium, and Uber that offer Error Prone extensions in the form of BugChecks. For example, Uber created a tool called NullAway to help

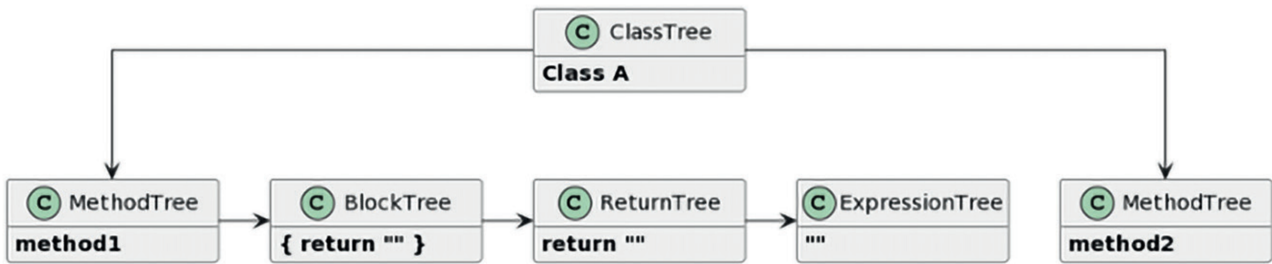


Image 2.

eliminate `NullPointerException`s. Using annotations, NullAway performs a fast type-based analysis to catch possible NPEs at compile-time.

{ LEVERAGING CUSTOM RULES }

Both Error Prone and Refaster offer the flexibility to create custom rules, proving invaluable in addressing various coding challenges. Whether it is updating deprecated APIs, handling common issues, refining code style, transitioning between libraries, or reducing technical debt, these rules can streamline the development process, saving time and manual effort.

At Picnic, we are avid users of Error Prone and Refaster. Over the years, we've created many additional rules to streamline our codebase. These rules are constantly applied across our 70+ Java repositories. They run as part of the build by default. This setup minimizes unproductive discussions about coding standards, bugs, or code style, enabling more focus on delivering value. Additionally, we allow developers to make contributions to our shared repository containing the refactoring rules, ensuring uniformity and ever-increasing code quality across all Java projects. As this consistency extends across teams, it has become easier for developers to switch between codebases and collaborate effectively.

{ CASE IN POINT: MIGRATING FROM TESTNG TO JUNIT + ASSERTJ }

Many years ago, Picnic decided to use TestNG as a testing framework. However, we later decided to use AssertJ for all assertions and JUnit as our testing framework. At that point, we had so much test code that performing this migration manually would be an enormous, slow process. Instead, we relied heavily on Error Prone and Refaster. First, we created many Refaster rules to migrate the assertions. Secondly, we created BugChecks to migrate the TestNG code to JUnit. Finally, we created BugChecks to suggest and apply best practices for JUnit code and make the code consistent and canonical. Instead of many engineers in different teams having to perform the migration, we had a small team write these rules and automate the whole migration for the rest of the company.

{ NEXT UP }

At Picnic, we've open-sourced our repository, Error Prone Support [3], where we maintain our Refaster rules, BugChecks, and modules that extend and improve the functionality of Error Prone itself. Next time, we'll discuss Error Prone Support, the main challenges we faced during its development, and rolling it out at Picnic, where it is now a core technology used by all our Java repositories. This brings some additional challenges, as introducing new rules can have a large impact, the new rules need to work for all codebases, and all developers need to agree with the changes. Stay tuned for the next installment, where you will learn how we created, and continue to manage, such a plugin in the Java ecosystem! <

{ REFERENCES }

1 https://errorprone.info
2 https://github.com/google/error-prone/issues/649
3 https://github.com/picnicSupermarket/error-prone-support

A PASSWORDLESS FUTURE

Passkeys for Developers

Passwords have been around for thousands of years and we were all happily sharing our Netflix passwords. They are probably not going anywhere but passwords are not the best way to secure our digital lives. They are a hassle for users and a security risk for applications. The future of secure authentication is passwordless and passkeys are leading the way.

Deepu K Sasidharan is a Java Champion working as a Staff Developer Advocate at Okta. He is the co-lead of JHipster and the creator of KDash and JDL Studio.

According to Verizon's 2023 Data Breach Investigations Report [1], stolen credentials and phishing account for over 65% of all data breaches.

Most of the password problems are human problems as they rely on us humans to remember them and to not share them and do a bunch of other stuff. The main problems with passwords are:

- Knowledge-based: People can be socially engineered to divulge passwords. If passwords are easy to remember they are also easy to guess. Sharing and reusing passwords makes them even more vulnerable.
- Phishing & Remote Replay: Phishing websites can easily harvest passwords from even the most tech-savvy. Accounts can be accessed remotely using harvested passwords.
- Data Breach: Applications become a target for data breaches when they store passwords.
- Password management: Passwords need password recovery and reset flows and multi-factor authentication flows to secure them further.

Did you know it could cost around 70$ to reset a password?

Of course, password managers help with some aspects of this and everyone should use one. But they are still an overhead and not convenient for everyone, especially non-tech folks.

{ PASSKEYS }

The obvious solution for the password problem is to go passwordless. This is where passkeys come into the picture.

"Passkeys are a password replacement that provides faster, easier, and more secure sign-ins to websites and apps across a user's devices. Unlike passwords, passkeys are resistant to phishing, are always strong, and are designed so that there are no shared secrets."— FIDO Alliance

A passkey is a unique cryptographic key pair that allows you to access online services without using passwords. It is based on asymmetric public-key cryptography.

Before we dive deep into passkeys let's look at some of the underlying technologies that make passkeys possible.

{ PUBLIC-KEY CRYPTOGRAPHY }

Asymmetric public key cryptography involves a pair of mathematically linked keys: a public key, which is shared openly, and a private key, kept secret by the owner.

This key pair can be used for encryption. The same key pair can also be used for digital signatures. A message signed with a private key can be verified with the public key, authenticating the sender's identity. This is what passkeys use (Image 1).

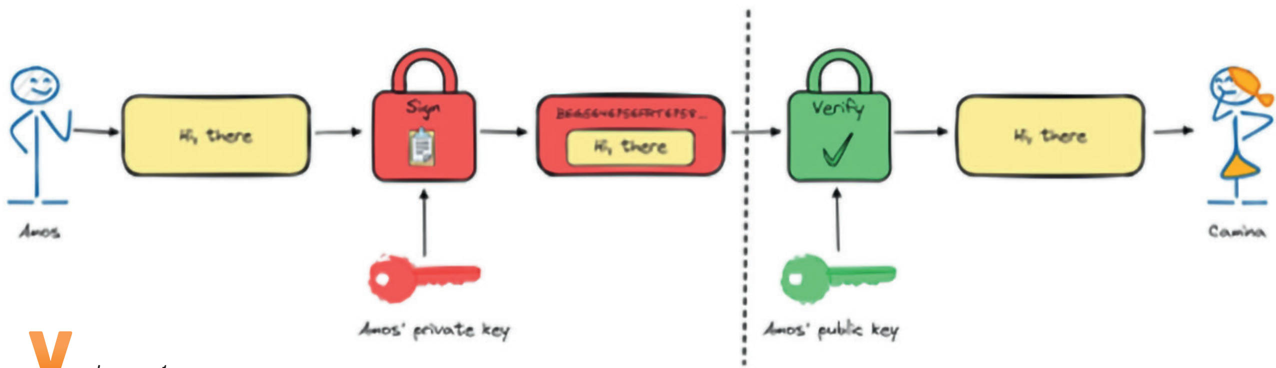


Image 1.

{ AUTHENTICATOR }

An authenticator is a hardware or software entity that can create and store public-private key pairs. There are two types of authenticators:

Platform authenticators: It is built into a user's device. For example, TouchID, smartphone authenticators, Windows Hello, and so on.
Roaming authenticators: A removable authenticator usable on any device. They are attached using USB, NFC, and/or Bluetooth. For example, security keys like YubiKey, Google Titan, and smartphones.

{ FIDO }

FIDO stands for Fast IDentity Online. It is a global authentication standard based on public key cryptography developed by the FIDO Alliance.

Passkeys are made possible by the FIDO2 [2] standard which is made up of Web Authentication (WebAuthn) and Client to Authenticator Protocol (CTAP).

{ WEB AUTHENTICATION }

Web Authentication is a W3C recommendation [3] that lets a webpage use a set of JavaScript APIs to talk to authenticators (Image 2).

The WebAuthn architecture consists of three main entities:

Authenticator: A device that lets a user authenticate by confirming their presence.
Relying Party: A server or an Identity Provider (IdP) that requires authentication. It issues challenges and stores public keys.
Client: A client consists of the user's browser. The client relays information between an authenticator and a relying party.

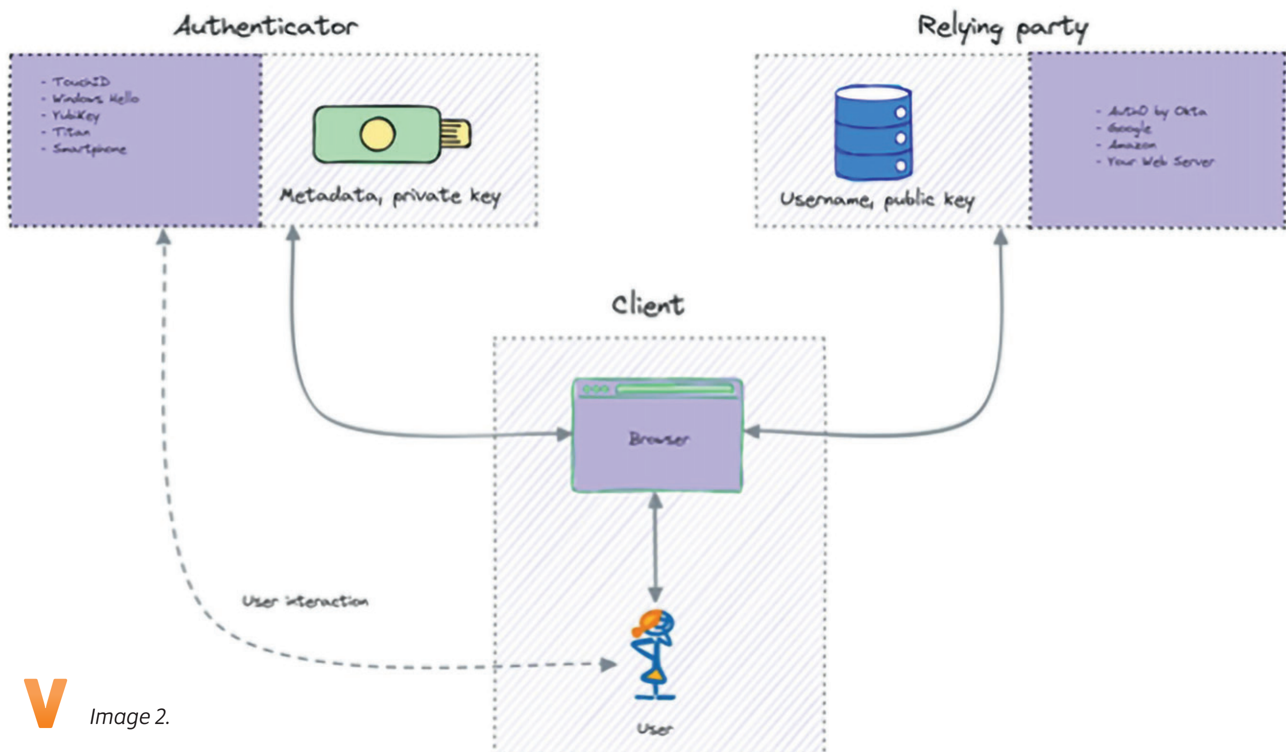


Image 2.

{ CLIENT TO AUTHENTICATOR PROTOCOL }

The FIDO Client to Authenticator Protocol [4] is used for communications with authenticators over a variety of transports like USB, NFC, and Bluetooth. It is used to send requests from WebAuthn to authenticators.

{ Types of passkeys }

Passkeys have two variants. Synced passkeys and device-bound passkeys.

Synced passkeys

Synced passkeys have a better user experience since the private keys are end-to-end encrypted and synced to the cloud. For example, on the Apple ecosystem, the private key is synced on your iCloud Keychain and you can register on one device and log in to any synced Apple device. The same goes for the Google ecosystem using the Chrome browser and Google Password Manager. Synced passkeys can be restored on new devices.

Device-bound passkeys

In the device-bound passkeys, the private key stays on the device itself and you need to authenticate using the same authenticator used for registration. It is slightly less convenient but more secure than synced passkeys. The relying party must support registering multiple credentials for a user as a backup.

{ HOW DOES IT WORK? }

Let's see how user registration and authentication work with passkeys.

User registration

First, let's see how the registration flow works (Image 3).

- The user begins registration. The relying party provides a randomly generated challenge string.
- The `navigator.credentials.create()` method of the WebAuthn API is invoked and the user provides approval using their authenticator.
- The authenticator creates a private-public key pair which is unique for the relying party's domain and the user. The private key is used to sign the challenge.
- The private key is stored on the authenticator.

For synced passkeys, the private key is also synced to a cloud service for backup and roaming.

- An attestation object is created which contains the public key, signed challenge, credential ID, and certificate.
- The attestation object and other metadata are passed to the relying party by the client-side implementation. The relying party verifies the signed challenge using the public key and registers the user by storing the public key and credential ID along with the user details.

User authentication

Now, let's see how the login flow works, which is quite similar except for the third step.

- The user begins the login flow. The relying party provides a randomly generated challenge string.
- The navigator.credentials.get() method of the WebAuthn API is invoked and the user provides approval using their authenticator.
- The authenticator retrieves the private keys for the relying party-'s domain name.

For synced passkeys, if the device is new, the private key is synced from a cloud service if available.

- The user selects the private key for their username. The private key is used to sign the challenge.
- An assertion object is created which contains the signed challenge and credential ID.
- The assertion object and other metadata are passed to the relying party by the client-side implementation. The relying party verifies the signed challenge using the public key stored for the user and authenticates the user.

{ WHY PASSKEYS? }

Passkeys are superior to password + traditional OTP MFA in terms of security and usability and they are as secure and more con-

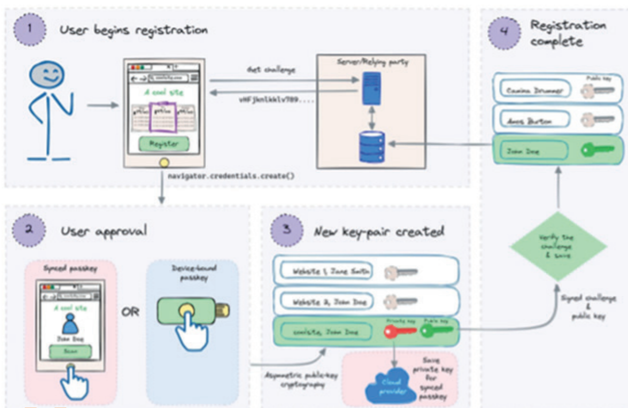


Image 3.

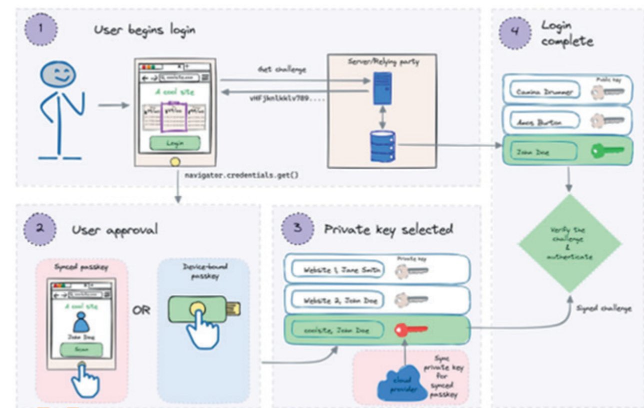


Image 4.

venient than password + FIDO MFA. Most importantly, you don't have to remember anything (Image 5).

Discoverable: Passwords are knowledge-based but passkeys are discoverable credentials and the browser can autofill them for a service making it unnecessary for you to remember even usernames. It doesn't rely on something you know, instead, it relies on something you have or something you are which is more secure from hacking and social engineering. They cannot be reused as they are unique per service and user combination and cannot be shared.
Phishing & Remote attack resistant: Passkeys cannot be phished as they rely on public key cryptography and are bound to the domain name of the website, making it impossible to work on a spoofed website. Passkeys rely on physical keys, like biometric sensors of platform authenticators or roaming authenticators like YubiKey, hence cannot be remotely breached.
Breach resistant: The website only stores the public key of a user which is useless to an attacker on a data breach on the server side. This makes the server less attractive to hackers.
Easier management: Synced passkeys are backed up and replicated across your devices by services like iCloud Keychain and Google Password Manager. This makes recovery part of the platform rather than the application.
Cross-device authentication: Passkeys can also perform cross-device authentication regardless of ecosystem or platform. For example, you can simply use your Android phone as an authenticator for your Apple laptop.

{ CHALLENGES }

There are still some challenges when it comes to passkeys:

OS/Browser support: It is dependent on the OS and Browser to implement the specs, and we all know how that turns out right? This means the support may not be uniform and can become fragmented. Browser and OS compatibility [5] is still catching up.
Cloud vendor reliance: It relies on companies like Apple, Google, and Microsoft to save the private keys in their cloud securely and protect them from breaches.
Enterprise use cases: Enterprise users might want more control and flexibility which could be a problem. For example, if an enterprise doesn't allow iCloud or Google Chrome on their computer, synced passkeys will not work there, only device-bound passkeys will work.
Reset & Recovery: There are no default recovery flows for device-bound passkeys, and applications might still need to implement recovery & reset flows to accommodate all use cases.

{ WEBAUTHN AND PASSKEYS FOR JAVA }

Though Web Authentication's user experience is a client-side implementation using JavaScript, the backend or Relying party can be a Java server. Ideally using an IdP like Auth0 would be the best option since it takes care of all the heavy lifting for you. You can find the sample app using Auth0 on GitHub [6].

~~Passwords~~	Passkeys
~~Phishing~~	Phishing resistant
~~Remote replay~~	Remote attack resistant
~~Data breach~~	Breach resistant
~~Reuse & share~~	Not Reusable & Shareable
~~Knowledge-based~~	Discoverable Credentials
~~Password management~~	Easier to maintain

Image 5.

But if you want to implement it yourself and walk the harder path, you can use one of the below libraries.

- WebAuthn4j [7]: A 100% FIDO2 conformant library with support for all attestation formats and validation. It is used by Keycloak and Spring Security.
- Java-webauthn-server [8]: A library from Yubico that supports many attestation format. But it is not 100% FIDO2 conformant.

You can find a sample app using WebAuthn4j on GitHub [9].

{ CONCLUSION }

Passkeys are the future of secure authentication. They are more secure and more convenient than passwords and traditional MFA. They are also more secure and more convenient than other passwordless methods like magic links, SMS/Email OTP, and push notifications. The wider adoption of passkeys could finally solve the password problem and make the internet a safer place. Learn more on learnpasskeys.io [10]. <

{ REFERENCES }

1 https://www.verizon.com/business/resources/reports/dbir/2023/summary-of-findings
2 https://fidoalliance.org/fido2/
3 https://www.w3.org/TR/webauthn/
4 https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html
5 https://passkeys.dev/device-support/
6 https://a0.to/jfall-passkey
7 https://github.com/webauthn4j/webauthn4j
8 https://github.com/Yubico/java-webauthn-server
9 https://a0.to/spring-webauthn
10 https://learnpasskeys.io/
11 https://deepu.tech/

'JE HOEFT ECHT GEEN ROCKET SCIENTIST TE ZIJN OM MEE TE DOEN'

Masters of Java 2024: nóg meer fun in funprogging

Op woensdag 6 november vindt Masters of Java 2024 plaats. Dit officieuze NK Java Programmeren wordt georganiseerd door NLJUG. First8 Conclusion is sinds lange tijd hoofdsponsor en verantwoordelijk voor de software en opdrachten. Presentator Ted Vinke: "We voegen steeds weer verrassende elementen aan de wedstrijd toe."

VOOR IEDEREEN MET JAVA-KENNIS

De afgelopen jaren zag Ted de Masters of Java steeds populairder worden. Dit jaar presenteert hij de wedstrijd samen met zijn First8-collega Sjoerd Hemels. Hij kijkt ontzettend uit naar de dag. "We zien steeds meer nieuwe gezichten die meedoen, als individu of samen met collega's. Het is dan ook een competitie waaraan iedereen met Java-kennis mee kan doen. Je hoeft echt geen Java-tovenaar te zijn", stelt Ted gerust. "Zie het als een uitstapje van je dagelijkse, vaak meer corporate werk. We zorgen ervoor dat de opdrachten fun zijn en je ontmoet veel leuke mensen."

FUN, MAAR OOK EEN FLINKE UITDAGING

De Java-experts van First8 Conclusion steken ieder jaar heel wat tijd en kennis in het doorontwikkelen van Masters of Java. Ted: "Daar halen we veel plezier uit! We proberen de puzzels ieder jaar weer uitdagender en grappiger te maken." Om daar met een knipoog aan toe te voegen: "Onze CTO Arjan Lamers bedenkt de opdrachten, dus het wordt sowieso brutal. Maar laat je daar vooral niet door afschrikken." Voor het oplossen van de puzzels krijgen kandidaten steeds dertig minuten, inclusief tests. "We willen niet dat iemand al na vijf minuten klaar is", vertelt Ted. "We zoeken steeds weer de sweet spot tussen fun en uitdaging. En per opdracht kun je meerdere punten verdienen, dus je blijft echt niet op nul staan."

MIDDELEEUWSE EDITOR

First8 Conclusion is ook verantwoordelijk voor de software die tijdens Masters of Java wordt gebruikt. "Je krijgt een laptop voor je met een soort Middeleeuwse editor", zegt Ted lachend. "Tijdens de wedstrijd blokkeren we de internetverbinding, zodat deelnemers alleen over de documentatie beschikken die wij ze overhandigen." Iedere editie van Masters of Java worden er weer leuke features toegevoegd. Zo bouwde First8 Conclusion-collega Suman Enait twee statistiekenpagina's, waarop standen en resultaten worden bijgehouden. Suman: "Er is een scherm met de realtime ranking en een ander scherm waarop we allerlei opvallende en grappige standen bijhouden. Denk aan de Cowboy Developer; de deelnemer die het minste test." Ted: "De features die Suman heeft toegevoegd, maken mijn rol als presentator nog leuker. De rankings bieden mij mooie haakjes om deelnemers of teams uit te lichten."

SCHRIJF JE GRATIS IN

Uit ervaring weet Ted dat het bij Masters of Java tijdens de opdrachten superspannend en heel stil in de ruimte is. "In de pauzes wordt er juist volop gemixt en gepraat. Hoe leuk is het om op deze manier samen met andere Java-liefhebbers in contact te komen? Wij hopen ook jou op 6 november te ontmoeten in Veenendaal!" Gratis inschrijven voor Masters of Java kan via NLJUG.org, onder het kopje Events.

FIRST8 CONCLUSION

WWW.CONCLUSION.NL/FIRST8

COLUMN

De ethiek van een ethische hacker

Variabelen zijn zelfstandige naamwoorden. *Klassenamen* zijn zelfstandige naamwoorden. *Functies* en *methodes* zijn werkwoorden die zo goed mogelijk omschrijven wat de functie/methode doet. Elke volwassen sector heeft zo zijn definities en best practices over taalgebruik. In de IT heeft Uncle Bob veel tijd en moeite geïnvesteerd om ons programmeurs uiteindelijk beter en professioneler te laten (samen-)werken.

Maja Reißner *is Docent cyber security.*

Ook als we over code praten gebruiken we weer vaktermen die duidelijke definities hebben, zodat we direct weten waar het over gaat. Zo is een functie een stuk code die input krijgt, deze verwerkt en weer output geeft. Een methode doet dit ook maar is daarnaast ook nog verbonden aan een object. En soms verandert de betekenis van woorden in de loop van de geschiedenis en wordt er voor gekozen om bepaalde termen aan te passen. Denk bijvoorbeeld aan de *master/slave* terminologie die nu is veranderd in *primary/second*.

En juist in deze context van IT en woorden, houdt me een specifieke term al langer bezig: de 'ethische hacker`. Voor mij is dit een dubbel ongelukkige term. Bij het woord hacker hebben de meeste mensen al een negatieve associatie en denken aan illegaal toegang verkrijgen, infiltreren of simpelweg stuk maken. En ben je dan zo ver dat iedereen in de groep begrijpt dat we daarmee iemand bedoelen die bestaande middelen gebruikt op een manier waarvoor ze niet bedoeld waren, bijvoorbeeld op een daarvoor niet bedoelde manier in een computernetwerk binnen te dringen, wat maakt dan deze hacker ethisch of niet ethisch?

In de cybersecurity bedoelen we met de term ethische hacker iemand die kwetsbaarheden in computersystemen zoekt en deze vervolgens niet openbaar maakt maar aan de betreffende bedrijven of overheid meldt zodat de kwetsbaarheden verholpen kunnen worden. Alleen heeft dat niet persé veel met ethiek te maken. En ook al is het idealiter legaal, dit haal je niet direct uit de term 'ethische hacker'. En dat vind ik problematisch. Er bestaan genoeg hackersgroepen (bijvoorbeeld *hacktivists* met extreme politieke ideologieën of mensen met een extreme religieuze overtuiging) die in hun eigen beleving ethisch zijn maar die in onze maatschappij niet ethisch en vooral ook niet legaal geacht worden. Deze groepen zoeken dan ook niet naar kwetsbaarheden om deze te melden, maar om hun eigen doelen na te streven - die zij dus zelf ethisch vinden en waardoor zij zichzelf wel als ethische hackers beschouwen.

Ik vind het problematisch dat we de term 'ethische hacker' gebruiken. Omdat ethiek los van de wet staat en ethiek als moraalfilosofie een eigen interpretatie voor je handelen toestaat. Ethiek kent zo gezien geen goede of slechte, maar éigen keuzes. Dit komt voor mij verdacht dicht bij de term "eigen rechter spelen" waarbij je ongeacht de wet zelf bepaalt wat wel of niet mag met je eigen interpretatie van ethiek.

Een betere term voor een persoon die legaal kwetsbaarheden in computersystemen zoekt om uiteindelijk de wereld een stuk beter te maken is naar mijn idee `*security tester*`. Deze titel voor zowel betaalde professionals als vrijwilligers is in mijn ogen voldoende. Zoals ook bij bijna elk ander beroep en functie is er geen noodzaak om woorden als ethisch of legaal in de titel te verwerken. Bij een arts gaan we er immers ook van uit dat het een ethisch persoon is die binnen het wettelijke kader werkt zonder het expliciet een `ethische arts` te noemen.

Wil je speels meer over ethiek leren? De *Absurd Trolley Problems* op https://neal.fun/absurd-trolley-problems/ zijn én hilarisch en confronterend als het gaat over hoe moeilijk morele keuzes kunnen zijn.

VAN HET BESTUUR

De winter is zo goed als voorbij en we kijken uit naar de lente. De dagen worden langer en persoonlijk kan ik niet wachten tot we weer regelmatig op het terras kunnen zitten en kunnen BBQ'en.

{ J-SPRING – 13 JUNI 2024 }

We hebben niet stilgezeten deze winter. Momenteel zijn we druk bezig met het samenstellen van het programma voor J-Spring. De meeste sprekers zijn al bevestigd en staan vermeld op onze website. Met een mooie, diverse line-up van sprekers en onderwerpen ben ik ervan overtuigd dat dit een editie wordt om naar uit te kijken. Net zoals voorgaande jaren vindt J-Spring plaats in de Jaarbeurs in Utrecht, ditmaal op donderdag 13 juni. Heb je nog geen tickets? Kijk dan snel op **jspring.nl** en zorg dat je deze editie niet mist.

{ BUSINESSPARTNER EN KENNISPARTNER MEETING }

De NLJUG kan niet zonder haar business- en kennispartners. Voor wie onbekend is met dit concept: businesspartners zijn de bedrijven die zijn gekoppeld aan de NLJUG en kennispartners zijn de onderwijsinstellingen. Speciaal voor deze groep organiseren we twee keer per jaar een meeting om iedereen up-to-date te houden. Onlangs hebben we een nieuwe draai gegeven aan de bijeenkomsten. Met een inspirerende talk van *Dominique Schreinemachers*, gevolgd door een speeddate-sessie, hebben we geprobeerd meer waarde te creëren voor onze partners. Het samenbrengen van business- en kennispartners is een speerpunt van de NLJUG, waarmee we de Java-community vanuit verschillende hoeken versterken. De positieve feedback op de laatste sessie beschouwen we als een groot succes.

{ BESTUURSZAKEN }

Er staan veranderingen op de agenda voor het bestuur. Helaas zal mijn collega *Noortje van Berlo* afscheid nemen. Aangezien Noortje een nieuwe onderneming heeft opgestart die wat verder afstaat van het Java-domein, heeft ze besloten haar bestuursfunctie neer te leggen. We zijn dus op zoek naar een nieuw bestuurslid. Meer informatie hierover vind je op onze website. Heb je interesse en ben je gepassioneerd om de Java-community in Nederland te ondersteunen? Meld je aan, we ontvangen graag je reactie (CV en motivatiebrief) via **board@nljug.org**.

{ TOT SLOT }

Enkele noemenswaardige evenementen die ik jullie niet wil onthouden. Zoals Teqnation op 22 mei in DeFabrique in Utrechten J-Fall op 7 november in Pathé Ede, met de pre-conference en de "Masters of Java" op 6 november.Na J-Spring en voorafgaand aan J-Fall starten we dit jaar ook opnieuw een Speaker Academy om jullie te ondersteunen bij het schrijven van abstracts en het geven van presentaties. En denk ook alvast na over welke innovatieve projecten je wilt nomineren voor de Innovation Awards.
Exacte data worden nog bekend gemaakt, dus houd onze nieuwsbrieven en social-media in de gaten.We hebben al met al dus veel mooie zaken op het programma staan en hopen dat jullie even enthousiast zijn als wij.

Brian Vermeer

FIRST8 | CONCLUSION

MASTERS OF JAVA 2024

Funprogging contest voor alle Java Developers

De Masters of Java is een roemruchte "funprogging contest" (gebaseerd op Java SE), die toegankelijk is voor iedere Java ontwikkelaar. In deze wedstrijd wordt niet de API-kennis, maar de echte programmeer-vaardigheid getest. In voorgaande edities heeft Masters of Java al heel wat verhitte strijd opgeleverd!

Er zijn 5 zeer diverse programmeeropdrachten voor de teams. Deze moeten binnen de tijdslimiet worden opgelost. Voor elke seconde die je overhoudt, scoor je een punt. Ook kun je extra punten scoren met bepaalde testen. Het team met de meeste punten wint.

Een team bestaat uit maximaal 2 ontwikkelaars en het team dat aan het einde van de dag de meeste punten heeft, mag zich "Master of Java 2024" noemen. Er zal naast eeuwige roem natuurlijk gestreden worden voor mooie en spectaculaire prijzen. De afgelopen jaren is de beslissing steeds gevallen tijdens de laatste opdracht. Het belooft dus een zinderende wedstrijd te worden!

Schrijf nu in voor Masters of Java

EEN NLJUG EVENT

Wanneer: woensdag 6 november
Tijd: 13.00u tot 17.30u
Waar: Van der Valk Hotel, Veenendaal
Kosten: Gratis

Als trotse hoofdsponsor zorgt First8 Conclusion elk jaar voor 5 uitdagende opdrachten en een robuuste game server. Met veel enthousiasme, bevlogenheid en passie werken wij aan deze en andere gave projecten. Iets voor jou? We maken graag kennis met je.

Benieuwd naar eerdere opdrachten?
Check https://github.com/First8/mastersofjava

Eventbrite

www.first8.nl

Aanmelden: https://nljug.org/masters-of-java-2024/

CONCLUSION | BUSINESS DONE DIFFERENTLY