Java Forum Stuttgart 2023

Modern Java

Ron Veen

Who is Ron Veen

- Java developer for 20+ years
- Special Agent @ Team Rockstars IT
- Writing a book about Migrating to cloud-native Jakarta EE

@ronveen

ronveen

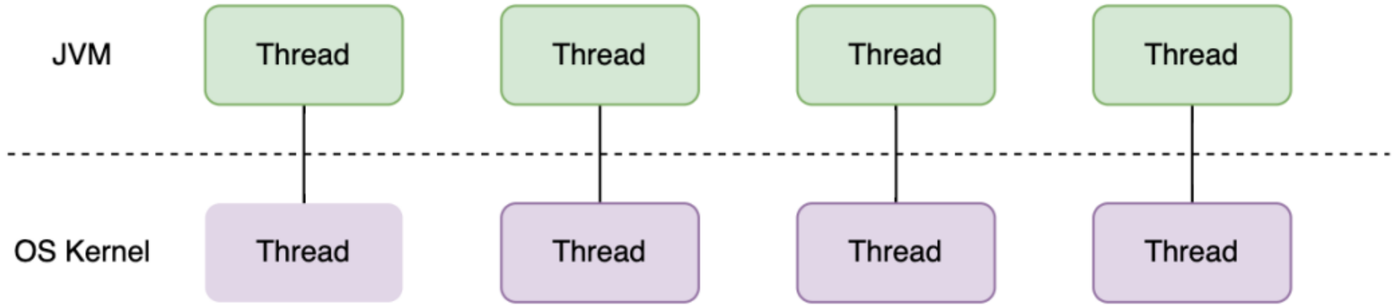
@ronveen@foojay.social

Agenda

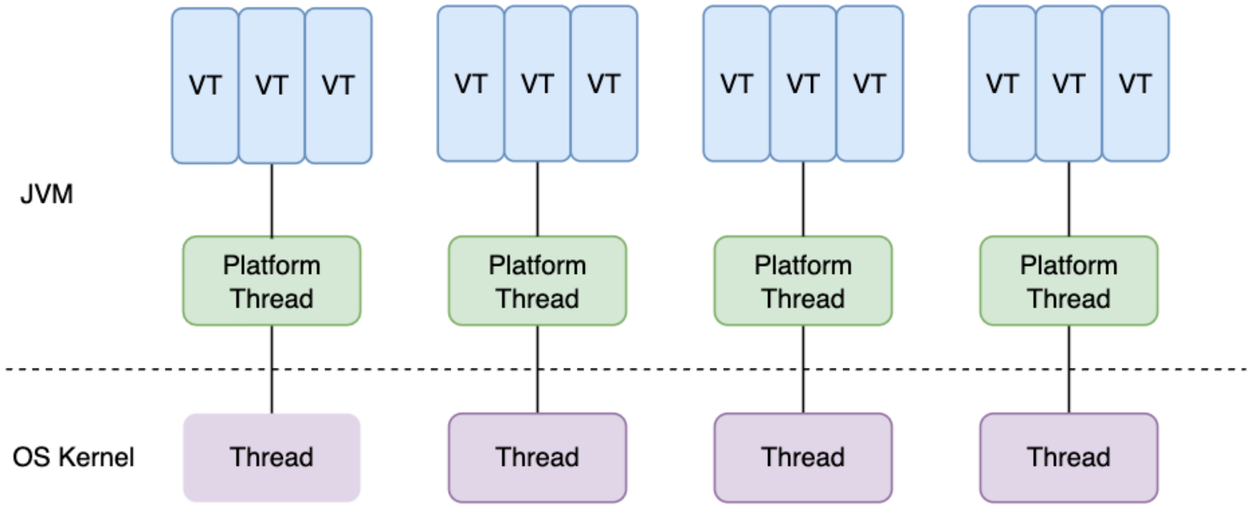
1. Virtual threads, structured concurrency, scoped values
2. What else is in Java 21
3. Records, sealed classes and pattern matching
4. Examples of data-oriented programming with Java

Part 1 - Virtual Threads, Structured Concurreny, Scoped Values

Java Threads



Virtual Threads



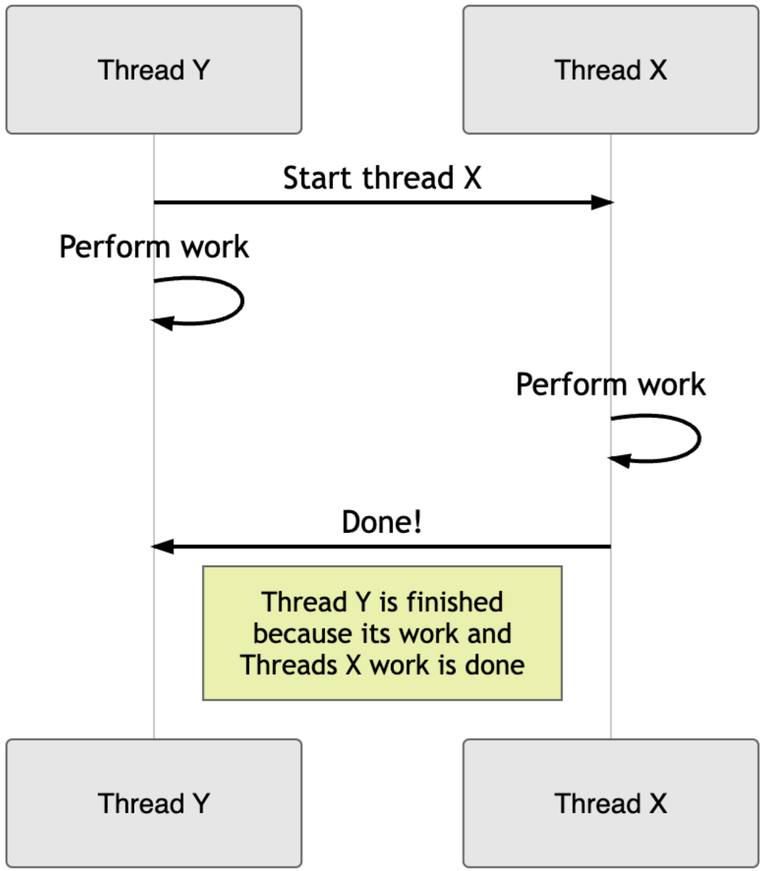
Factory methods

```
public class VirtualThread {
    Runnable runnable = () -> {

        System.out.println("I am a virtual thread: " +      Thread.currentThread().isVirtual());
    };

    public static void main(String... args) {
        new VirtualThread().run();
    }

    private void run() {
        Thread.ofVirtual().start(runnable);
        Thread.ofPlatform().start(runnable);
    }

}
```

Virtual Thread Per Task Executor

```
private void run() {
    try(var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        executor.submit(runnable);
    }
}
```

Structured Concurrency



Invoke All pattern

```
String getDog() throws ExecutionException, InterruptedException {

    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        Task<String>  name  = scope.fork(this::getName);
        Task<String> breed = scope.fork(this::dogBreed);

        scope.join();
        scope.throwIfFailed();

        return "it's name is:" + name.resultNow() + ", and is a " + breed.resultNow();
    }
}
```

Invoke Any pattern

```
String getDog() throws ExecutionException, InterruptedException {

        try (var scope = new StructuredTaskScope.ShutdownOnSuccess<String>()) {

        scope.fork(this::getName);
        scope.fork(this::dogBreed);

        scope.join();

        return "result: " + scope.result();
    }
```

Thread Locals

1. Share information between different components of your application
2. Create a ThreadLocal instance that is reachable from anywhere
3. You can provide an initial value on creation
4. Or set a value using set (T value)
5. Retrieve the value anywhere via get( )

The problem with Thread Locals

1. Mutable
2. Resource intensive
3. Leaking

Enter Scoped Values

1. Exist for a limited time (lifetime of the Runnable)
2. Only the thread that wrote the value can read it
3. Immutable
4. Passed by reference

Scoped Values


```java
public class AdvancedScopedExample {
    public static final ScopedValue<String> SCOPED_SECRET = ScopedValue.newInstance();
    public static final ScopedValue<String> SCOPED_USER = ScopedValue.newInstance();

    Runnable runnable = () ->
        System.out.println("The secret is: " + (SCOPED_SECRET.isBound() ? SCOPED_SECRET.get() : "<unknown!!>"));


    public void shareIt() {
        String secret = "rvkaj9893juf9qh39";
        runnable.run();
        ScopedValue
                .where(SCOPED_SECRET, secret)
                .where(SCOPED_USER, "admin")
                .run(runnable);
        runnable.run();
    }

    public static void main(String... args) {
        new AdvancedScopedExample().shareIt();
    }


}
```

Part 2 - What else is in Java 21

Agenda

1. Simpler main methods
2. Unnamed classes
3. Unnamed variables
4. Sequenced Collections
5. String Templates

Simpler main methods

```java
public static void main(String... args) {
    System.out.println("static void main(String... args)");
}

static void main(String... args) {
    System.out.println("static void main(String... args)");
}

static void main() {
    System.out.println("static void main()");
}

void main() {
    System.out.println("void main()");
}
```

Unnamed classes

```java
void main() {
    System.out.println("Started from an unnamed class");
}
```

Unnamed variables

```
int count = 0;
for (String _ : fruits) {
    System.out.println("This is iteration: " + ++count);
}


//  Ignoring the return value of a method
var _ = fruits.getFirst();



} catch (Exception _) {
    System.out.println("Division by zero");
}
```

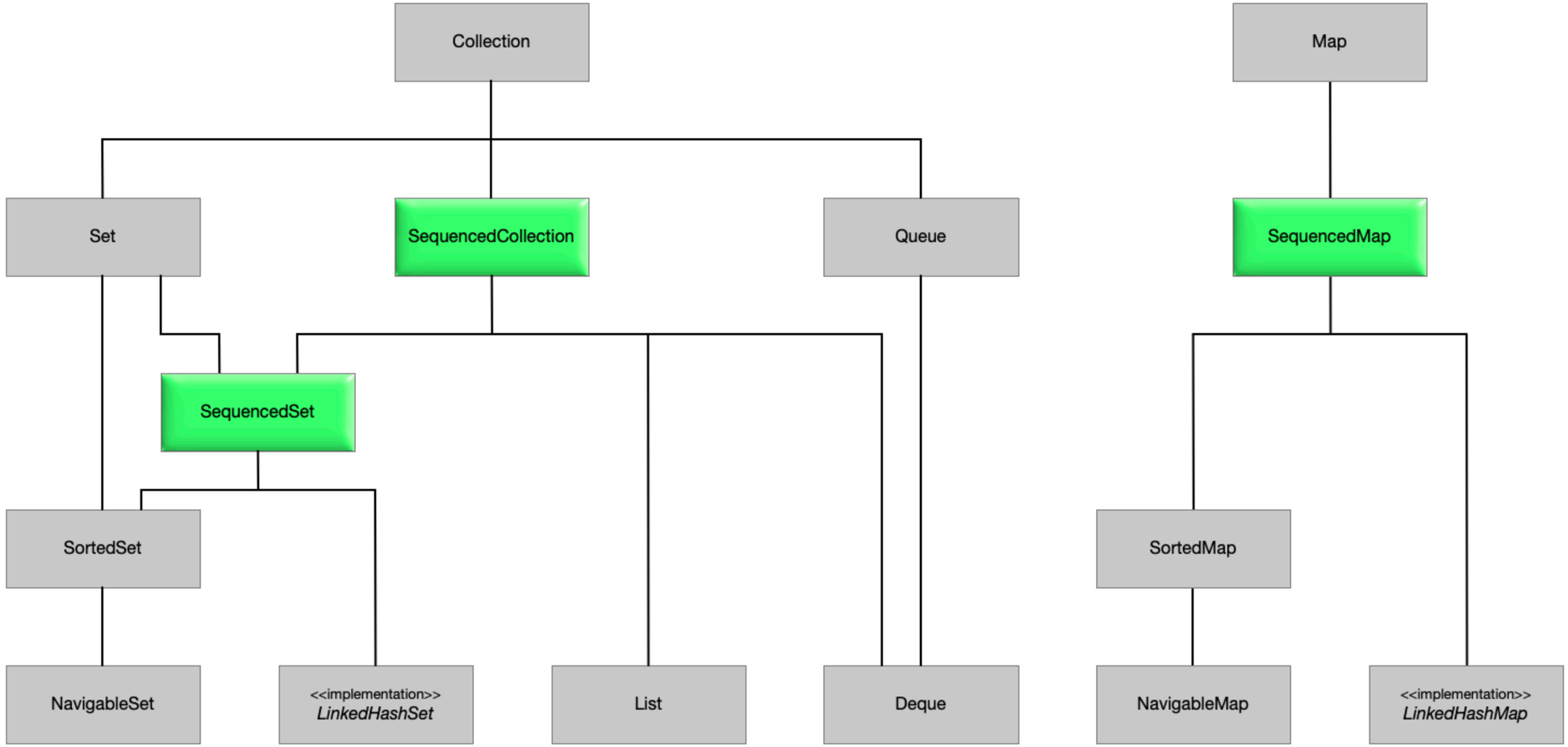
Unnamed variables

```java
public record Country(String name, String continent, int population) {

    public int worldPopulation(List<Country> countries) {
        int total = 0;
        for (Country(var _, var _, int p)  : countries) {
            total += p;
        }
        return total;
    }
}
```

Unnamed variables

```java
public record Country(String name, String continent, int population) {

    public int worldPopulation(List<Country> countries) {
        int total = 0;

        for (var country : countries) {
            total += switch (country) {
                case Country(_, _, int p) -> p;
            };

        }
        return total;
    }

}
```

Sequenced Collections



Sequenced Collections

```java
public class SequencedCollections {
    ArrayList<String> list = new ArrayList<>(List.of("Rot", "Schwarz", "Gold"));

    public void someMethod() {
        list.getFirst();
        list.getLast();
        list.addFirst( element: "First now");
        list.addLast( element: "Last now");
        list.removeFirst();
        list.removeLast();
        list.reversed();
    }

    void main() {
        //  Prints Gold, Schwarz, Rot
        list.reversed().forEach(System.out::println);
    }
}
```

String templates

```
void main() {
    String conference = "Java Forum Stuttgart";
    int times = 26;

    String info = STR."Dies ist die \{times}. Ausgabe von \{conference}";

    System.out.println(info);
}
```

```
@FunctionalInterface
public interface Processor<R, E extends Throwable> {

    /**
     * Constructs a result based on the template fragments and values in the
     * supplied {@link StringTemplate stringTemplate} object.
     * @param stringTemplate  a {@link StringTemplate} instance
     *
     * @return constructed object of type R
     *
     * @throws E exception thrown by the template processor when validation fails
     */
    R process(StringTemplate stringTemplate) throws E;
```

String templates

```
record QueryBuilder(Connection conn)
        implements StringTemplate.Processor<PreparedStatement, SQLException> {

    public PreparedStatement process(StringTemplate st) throws SQLException {
        // 1. Replace StringTemplate placeholders with PreparedStatement placeholders
        String query = String.join( delimiter: "?", st.fragments());

        // 2. Create the PreparedStatement on the connection
        PreparedStatement ps = conn.prepareStatement(query);

        // 3. Set parameters of the PreparedStatement
        int index = 1;
        for (Object value : st.values()) {
            switch (value) {
                case Integer i -> ps.setInt(index++, i);
                case Float f   -> ps.setFloat(index++, f);
                case Double d  -> ps.setDouble(index++, d);
                case Boolean b -> ps.setBoolean(index++, b);
                default        -> ps.setString(index++, String.valueOf(value));
            }
        }

        return ps;
    }
}
```

String templates

```
var DB = new QueryBuilder(conn);
PreparedStatement ps = DB."""
    SELECT *
    FROM Person p
    WHERE p.last_name = \{name}
    ORDER BY p.last_name DESC""";
ResultSet rs = ps.executeQuery();
```

Part 3 - Records, sealed classes and pattern matching

Modern Java support for data-oriented programming

- Records
- Sealed classes
- Pattern matching
- Switch functions for pattern matching

Records

```java
public record UserRecord(String email, String password, boolean isBlocked, int loginAttemps) {
}
```

```java
var user = new UserRecord( email: "ronveen@geecon.pl",  password: "DoNotTell!",  isBlocked: false,  loginAttemps: 0);
```

```java
var email = user.email();
```

```java
public UserRecord {
    requireNonNull(email,  message: "email cannot be null");
    requireNonNull(password,  message: "password cannot be nul
    loginAttemps = (loginAttemps < 0 ? 0 : loginAttemps);
}
```

Sealed classes

```
sealed interface User permits Approver, ProjectManager, RegularUser{
}
```

```
public sealed class Approver implements User permits TimesheetApprover, InvoiceApprover {
}
```

```
public final class ProjectManager implements User {
}
```

```
public non-sealed class RegularUser implements User {
}
```

Pattern Matching

- Predicate
- Object to test against
- Pattern variables
- Flow scope

Pattern Matching: Instanceof

```java
public static boolean allowApproving(User user) {
    return user instanceof Approver a;
}
```

- Predicate ==> instanceof Approver
- Object to test against ==> user
- Pattern variables ==> a
- Flow scope ==> between { }

Pattern Matching: switch

```java
public static boolean allowApproving(User user) {
    return switch (user) {
        case Approver a -> true;
        case ProjectManager p -> false;
        case RegularUser r -> false;
    };
}
```

```java
sealed interface User permits Approver, ProjectManager, RegularUser{
}
```

```java
public static boolean allowApproving(User user) {
    return switch (user) {
        case Approver a when a.isActive() -> true;
        case Approver a -> false;
        case ProjectManager p -> false;
        case RegularUser r -> false;
    };
}
```

Pattern Matching: record

```
if (user instanceof UserRecord ur) {
    return ur.isBlocked();
}
```

```
private static void rpmv(Record record) {
    switch (record) {

        case null -> System.out.println("null");

        case UserRecord(var e, var p, var b, var a) ->
                System.out.println("User " + e + ", blocked=" + b);

        default -> { }
    }
}
```

```
 public static void sendMailVar(List<UserRecord> users) {

     for (UserRecord(var e, var p, var b, var f) : users) {
         if (!b)  sendUserMail(e);
     }

 }
```

Part 4 - Combining Java and data-oriented programming

Data-oriented programming

JAVA

QCon New York (June 13-15): Le

Data Oriented Programming in Java

LIKE 7

JUN 20, 2022 • 22 MIN READ

by
Brian Goetz FOLLOW
Java Language Architect at Oracle.
reviewed by
Daniel Bryant FOLLOW
DevRel @ Ambassador Labs | InfoQ News Manager | QCon PC

Key Takeaways

- Project Amber has brought a number of new features to Java in recent years. While each of these features are self-contained, they are also d… together. Specifically, records, sealed classes, and pattern mat… together to enable easier data-oriented programming in Java.
- OOP encourages us to model complex entities and processes u… which combine state and behavior. OOP is at its best when it is… defending boundaries.
- Java's strong static typing and class-based modeling can still b… useful for smaller programs, just in different ways.
- Data-oriented programming encourages us to model data as (i… and keep the code that embodies the business logic of how we… separately. Records, sealed classes, and pattern matching, ma…
- When we're modeling complex entities, OO techniques have a … when we're modeling simple services that process plain, ad-ho… techniques of data-oriented programming may offer us a straig…

…OOP and data-oriented programming are no… …ifferent granularities and situations. We can… …see fit.

• • •

…ght a number of new features to Java in rec… …text blocks, records, sealed classes, patter… …se features are self-contained, they are als… …ecords, sealed classes, and pattern matchin… …*nted programming* in Java. In this article, we… …how it might affect how we program in Java…

Principles of data-oriented programming

- Separate data from logic
- Data is stored in generic data structures
- Data is immutable

Alternative optional

```
sealed interface Opt<T> {
    record Some<T>(T value) implements Opt<T> { }
    record None<T>() implements Opt<T> { }
}
```

```
private Opt<String> validate(String... args) {
    if (args.length == 0) {
        return new Opt.None<>();
    } else {
        return new Opt.Some<>(args[0]);
    }
}
```

```
<T> void process(Opt<T> opt) {
    switch (opt) {
        case Some<T>(var v) -> System.out.println("v = " + v);
        case None<T>() -> System.out.println("It's nothing");
    };
}
```

Usage of data-oriented programming

```java
public sealed interface SearchResult<T> {
    record NoResult<T>() implements SearchResult<T> { }
    record ExactResult<T>(T result) implements SearchResult<T> { }
    record MultiResult<T>(List<T> result, int totalCount) implements SearchResult<T> { }
}
```

```java
public SearchResult<Project> findProjects(String searchKey) {
    return new SearchResult.ExactResult<>(new Project(UUID.randomUUID(), searchKey,Status.Active, List.of()));
}
```

```java
public Object search(String value) {
    return switch (service.findProjects(value)) {
        case SearchResult.NoResult() -> noResultsFound();
        case SearchResult.ExactResult(var p) -> showDetails(p);
        case SearchResult.MultiResult(var p, var count) -> showSelection(p, count);
    };
}
```

Summary

- Data-oriented programming treats data as first-class citizens
- Data drives your application
- Java supports data-oriented programming via records, sealed classes and pattern matching

Questions