Java Moderno em 30 mins

Eder Ignatowicz
Principal Software Engineer
@ederign

Red Hat

O que eu acho mais maneiro do Java moderno :)

JEP 358: Helpful NullPointerExceptions

```
a.i = 99;

Exception in thread "main"
java.lang.NullPointerException
    at Prog.main(Prog.java:5)
```

JDK 14

```
Exception in thread "main"
java.lang.NullPointerException:
        Cannot assign field "i" because "a" is null
    at Prog.main(Prog.java:5)
```

JEP 355: Text Blocks

```
String sql = "SELECT COUNT(*) FROM table; -- Use this to determine rand_low and rand_high\n" +
             "\n" +
             "  SELECT *\n" +
             "    FROM table\n" +
             "   WHERE frozen_rand BETWEEN %(rand_low)s AND %(rand_high)s\n" +
             "ORDER BY RAND() LIMIT 1000";
```

```
var sql = """
                SELECT COUNT(*) FROM table; -- Use this to determine rand_low and rand_high

                  SELECT *
                    FROM table
                   WHERE frozen_rand BETWEEN %(rand_low)s AND %(rand_high)s
                ORDER BY RAND() LIMIT 1000
          """;
```

JEP 286: Local-Variable Type Inference

```
List<Student> students = new ArrayList<>();
students.removeIf(s -> s.getId() == desiredId);

var foo = 1;

var bestStudent = new Student("Dora");

for (var student: students) { /* … */ }

for (var i = 0; i < 10; i++) { /* … */ }
```

```
var x = y.bar();
```

```
Map<Long, Student> idToStudent = studentsRepository.getStudentId();
List<Student> enrolledStudents = studentsRepository.getEnrolledStudents();
Address addressOfBestStudent = studentsRepository.getAddress(bestStudent);

var idToStudent = studentsRepository.getStudentId();
var enrolledStudents = studentsRepository.getEnrolledStudents();
var addressOfTopStudent =  studentsRepository.getAddress(bestStudent);
```

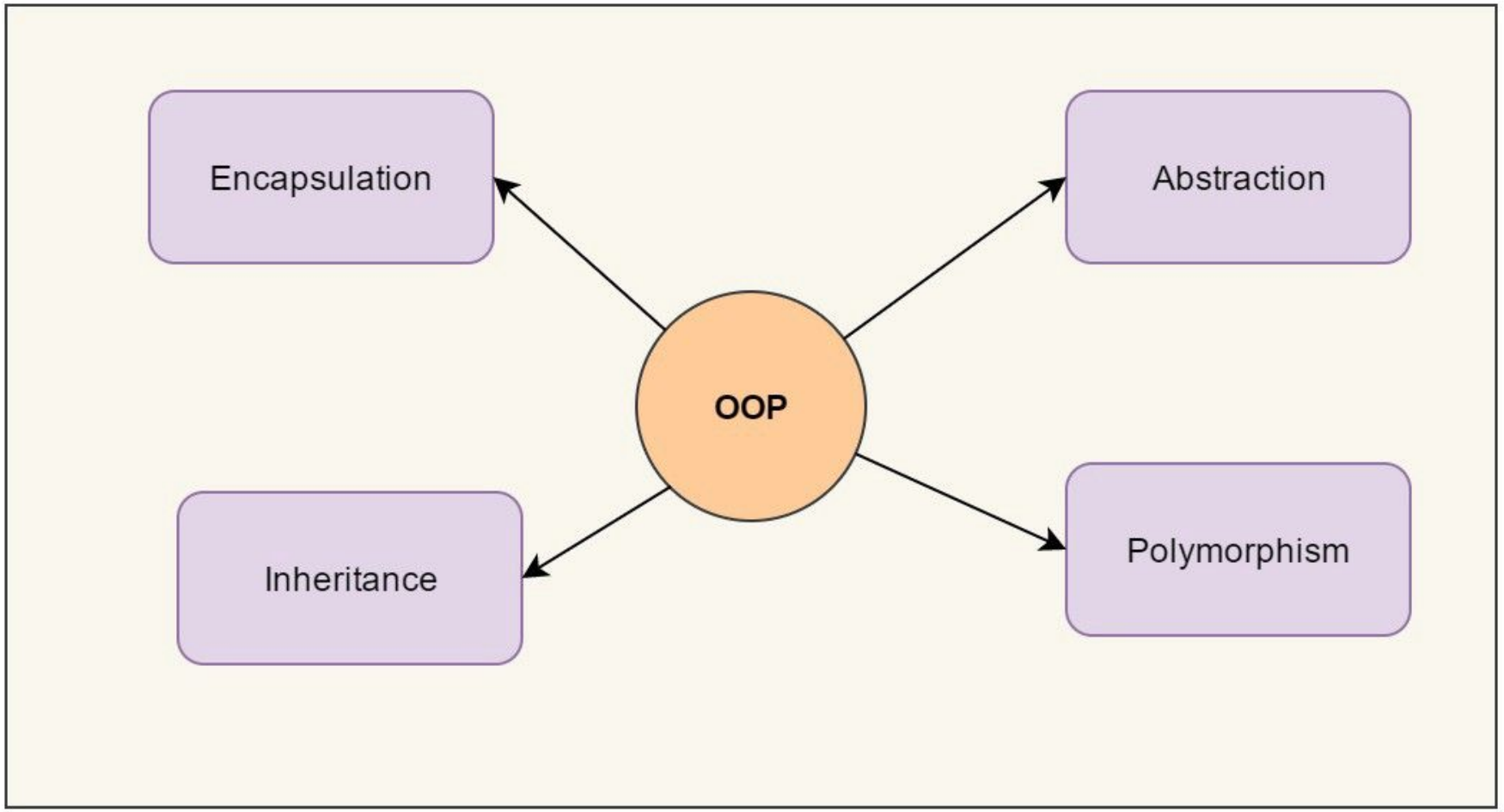
JEP 361: Switch Expressions (Standard)

```
switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numLetters = 6;
        break;
    case TUESDAY:
        numLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numLetters = 8;
        break;
    case WEDNESDAY:
        numLetters = 9;
        break;
}
```

```
switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> numLetters = 6;
    case TUESDAY                -> numLetters = 7;
    case THURSDAY, SATURDAY     -> numLetters = 8;
    case WEDNESDAY              -> numLetters = 9;
}
```

```
int numberOfDays = switch (day) {
    case FRIDAY, SUNDAY -> 6;
    case TUESDAY -> 7;
    case THURSDAY, SATURDAY -> 8;
    default -> {
        if (day == Days.WEDNESDAY) {
            yield 9;
        }
        else{
            yield -1;
        }
    }
};
```

JEP 359: Records (Preview)

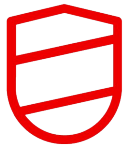


Four Pillars of Object Oriented Programming

Architectures

Evolutionary

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.



Microservices

Architectural style that structures an application as a collection of independent services.

Serverless

Incorporate third-party “Backend as a Service”, and/or that include custom code run as Functions.

Micro Frontends

Design approach in which a front-end app is decomposed into individual, semi-independent “microapps” working loosely together.

```java
package me.ederign;

public class SampleTask {

    private long id;
    private long owner;
    private String fieldA;
    private String fieldB;
    private String fieldC;
    private String fieldD;

...

}
```

```java
package me.ederign;

public class SampleTask {

    private long id;
    private long owner;
    private String fieldA;
    private String fieldB;
    private String fieldC;
    private String fieldD;

    public SampleTask(long id, long owner, String fieldA,
String fieldB, String fieldC, String fieldD) {
        this.id = id;
        this.owner = owner;
        this.fieldA = fieldA;
        this.fieldB = fieldB;
        this.fieldC = fieldC;
        this.fieldD = fieldD;
    }
}
```

```java
package me.ederign;

import java.util.Objects;

public class SampleTask {

    private long id;
    private long owner;
    private String fieldA;
    private String fieldB;
    private String fieldC;
    private String fieldD;

    public SampleTask(long id, long owner, String fieldA, String fieldB, String fieldC, String fieldD) {
        this.id = id;
        this.owner = owner;
        this.fieldA = fieldA;
        this.fieldB = fieldB;
        this.fieldC = fieldC;
        this.fieldD = fieldD;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SampleTask that = (SampleTask) o;
        return id == that.id &&
                owner == that.owner &&
                Objects.equals(fieldA, that.fieldA) &&
                Objects.equals(fieldB, that.fieldB) &&
                Objects.equals(fieldC, that.fieldC) &&
                Objects.equals(fieldD, that.fieldD);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, owner, fieldA, fieldB, fieldC, fieldD);
    }
}
```

88 LINHAS!

```java
package me.ederign;

import java.util.Objects;

public class SampleTask {

    private long id;
    private long owner;
    private String fieldA;
    private String fieldB;
    private String fieldC;
    private String fieldD;

    public SampleTask(long id, long owner, String fieldA, String fieldB, String fieldC, String fieldD) {
        this.id = id;
        this.owner = owner;
        this.fieldA = fieldA;
        this.fieldB = fieldB;
        this.fieldC = fieldC;
        this.fieldD = fieldD;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        SampleTask that = (SampleTask) o;
        return id == that.id &&
                owner == that.owner &&
                Objects.equals(fieldA, that.fieldA) &&
                Objects.equals(fieldB, that.fieldB) &&
                Objects.equals(fieldC, that.fieldC) &&
                Objects.equals(fieldD, that.fieldD);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, owner, fieldA, fieldB, fieldC, fieldD);
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getOwner() {
        return owner;
    }

    public void setOwner(long owner) {
        this.owner = owner;
    }

    public String getFieldA() {
        return fieldA;
    }

    public void setFieldA(String fieldA) {
        this.fieldA = fieldA;
    }

    public String getFieldB() {
        return fieldB;
    }

    public void setFieldB(String fieldB) {
        this.fieldB = fieldB;
    }

    public String getFieldC() {
        return fieldC;
    }

    public void setFieldC(String fieldC) {
        this.fieldC = fieldC;
    }

    public String getFieldD() {
        return fieldD;
    }

    public void setFieldD(String fieldD) {
        this.fieldD = fieldD;
```


Red Hat

```java
package me.ederign;

public class SampleTask {

    private long id;
    private long owner;
    private String fieldA;
    private String fieldB;
    private String fieldC;
    private String fieldD;

...

}
```

```
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

```java
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

Fields imutáveis

Constructors

equals, hashCode and toString

```
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

"plain data" aggregate

(DTO, wrapper, transfer objects,

etc)

```
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

Desacoplamento total para

data classes entre o estado e a sua

API

```
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

Fit natural para externalização segura em sistemas distribuídos (serialização, marshalling para JSON/XML, mapping)

```java
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

Aceita:

Novos construtores (até o canonico) com lógica adicional

Static fields/methods

Implementa interfaces

Annotations

```
 // IntelliJ API Decompiler stub source generated from a class file
  // Implementation of methods is not available

package me.ederign;

public final class SampleTask extends java.lang.Record {
    private final long id;
    private final long owner;
    private final java.lang.String fieldA;
    private final java.lang.String fieldB;
    private final java.lang.String fieldC;
    private final java.lang.String fieldD;

    public SampleTask(long id, long owner, java.lang.String fieldA, java.lang.String fieldB, java.lang.String
fieldC, java.lang.String fieldD) { /* compiled code */ }

    public long id() { /* compiled code */ }

    public long owner() { /* compiled code */ }

    public java.lang.String fieldA() { /* compiled code */ }

    public java.lang.String fieldB() { /* compiled code */ }

    public java.lang.String fieldC() { /* compiled code */ }

    public java.lang.String fieldD() { /* compiled code */ }

    public java.lang.String toString() { /* compiled code */ }

    public final int hashCode() { /* compiled code */ }

    public final boolean equals(java.lang.Object o) { /* compiled code */ }
}
```

```java
public record SampleTask(long id,
                         long owner,
                         String fieldA,
                         String fieldB,
                         String fieldC,
                         String fieldD) {}
```

"plain data" aggregate

Fit perfeito para

Arquiteturas Distribuídas

JEP 360: Sealed Classes (Preview)

```
sealed interface Shape
        permits Circle, Rectangle {
}

record Circle(Point center, int radius) implements Shape { }

record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
```

```
sealed interface Shape
        permits Circle, Rectangle {
}

record Circle(Point center, int radius) implements Shape { }

record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
```

Vantagens

Designer da API controla melhor as implementações

O compilador pode inferir mais coisas...

Sealed Classes + Records

```
sealed interface Shape
        permits Circle, Rectangle {
}

record Circle(Point center, int radius) implements Shape { }

record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
```

Sealed Classes ~= 'Sum Types'

O conjunto de todos os tipos Shape s é igual ao conjunto

de todos os Circle c mais todos os Rectable S

```
sealed interface Shape
        permits Circle, Rectangle {
}

record Circle(Point center, int radius) implements Shape { }

record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
```

Record ~= 'Product Types'

Todos os possíveis estados (state space) é um subconjunto do produto cartesianos de todos seus componentes.

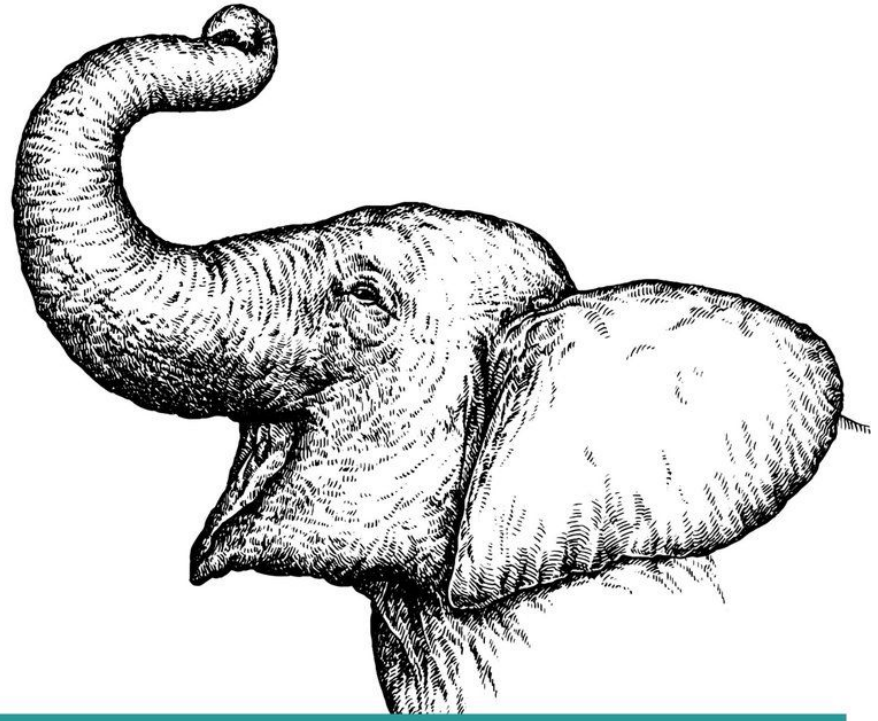
```
sealed interface Shape
        permits Circle, Rectangle {
}

record Circle(Point center, int radius) implements Shape { }

record Rectangle(Point lowerLeft, Point upperRight) implements Shape { }
```

Code smell???

Isto não viola o encapsulamento?

Pq o Java tá fazendo isto?

The answer to every programming question ever conceived



It Depends

The Definitive Guide

O RLY?

@ThePracticalDev

Red Hat

"Sealed classes work together with records and pattern matching to support a more data-centric form of programming."
Brian Goetz

JDK 15+

Sealed Classes + Records

JEP 305/JEP 375: Pattern Matching for instanceof

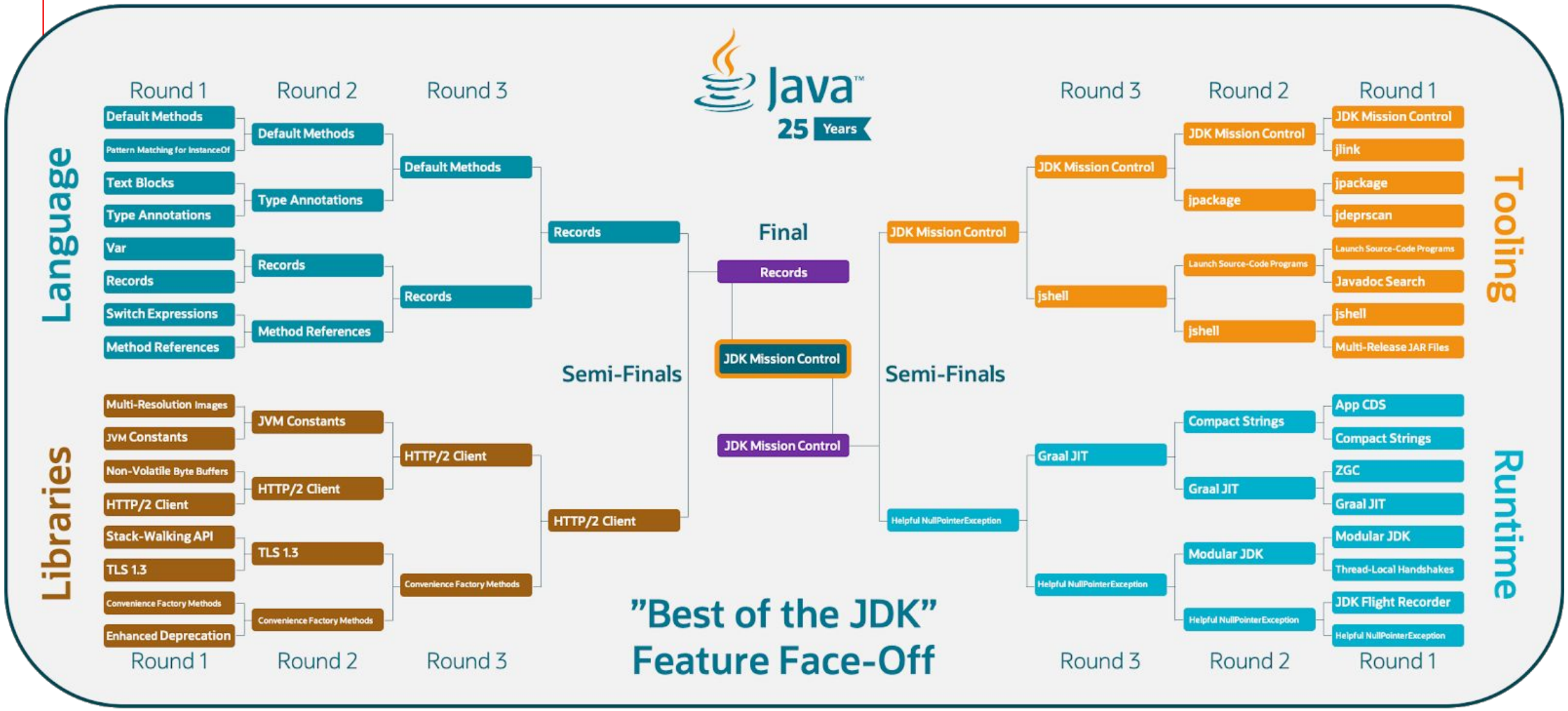
```
static int getCenter(Shape shape) {
    if (shape instanceof Rectangle) {
        return ((Rectangle) shape).upperRight().x;
    } else if (shape instanceof Circle) {
        return ((Circle) shape).radius();
    }
    return -1;
}
```

```
static int getCenterJ15(Shape shape) {
        if (shape instanceof Rectangle r) {
            return r.upperRight().x;
        } else if (shape instanceof  Circle c) {
            return c.radius();
        }
        return -1;
    }
```

JDK 15+

JEP draft: Pattern matching for switch (Preview)

```
float area = switch (shape) {
    case Circle c -> Math.PI * c.radius() * c.radius();
    case Rectangle r -> Math.abs((r.upperRight().y() - r.lowerLeft().y())
                                 * (r.upperRight().x() - r.lowerLeft().x()));
    // no default needed!
}
```


Java™
25 Years
Language
Libraries
Tooling
Runtime
Round 1
Round 2
Round 3
Default Methods
Pattern Matching for InstanceOf
Text Blocks
Type Annotations
Var
Records
Switch Expressions
Method References
Multi-Resolution Images
JVM Constants
Non-Volatile Byte Buffers
HTTP/2 Client
Stack-Walking API
TLS 1.3
Convenience Factory Methods
Enhanced Deprecation
Semi-Finals
Final
JDK Mission Control
jlink
jpackage
jdeprscan
Launch Source-Code Programs
Javadoc Search
jshell
Multi-Release JAR Files
App CDS
Compact Strings
ZGC
Graal JIT
Modular JDK
Thread-Local Handshakes
JDK Flight Recorder
Helpful NullPointerException
"Best of the JDK"
Feature Face-Off

Red Hat

Thank you

Eder Ignatowicz.

Principal Software Engineer

@ederign

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat