Una mirada 👀 funcional a Java 2️⃣2️⃣

Antonio Muñoz

¿Quien soy?

- Programo en Java desde Java 1.1.
- Actualmente trabajo en https://clarity.ai como backender.
- Mi primera charla en una conferencia.
- Me encontraréis en:
    - Mastodon: @tonivade@jvm.social
    - Github: https://github.com/tonivade
    - Email: me@tonivade.es

Encuesta

- Java 22? 😎
- Java 21? 👍
- Java 17? 🆗
- Java 11? ⚠️
- Todavía Java 8? 😢
- Y anteriores a Java 8? 🤯

Jetbrains ecosystem survey

- ¿Qué versión de Java usáis regularmente?
- El 11% usan Java 20.
- El 45% usan Java 17.
- El 38% usan Java 11.
- Todavía el 50% usan Java 8.

Java 8️⃣ 👴

- Lanzado en Marzo 2014 🚀
- No existía ni tiktok 🎶 ni openai 🤖
- Ese mismo año se lanzó spring-boot v1.0.
- Docker y kubernetes daban sus primeros pasos 👶

Agenda 📅

- El largo camino a Java 22.
- Tipos de datos algebraicos.
- Ejemplos.
- Futuro.

Java Release Cadence ☕

- Dos release al año.
- LTS.
- Preview features.
- They have a plan.

Switch expressions 🛤️

```java
var value = switch (input) {
    case "a" -> 1;
    case "b" -> 2;
    default -> 0;
};
```

- Incluido en Java 14.
- Una nueva vida para `switch`.
- Expresión.
- `yield`.

yield

```
var value = switch (input) {
    case "a" -> {
        yield 1;
    }
    case "b" -> {
        yield 2;
    }
    default -> {
        yield 0;
    }
};
```

Records 📽️

```
public record Movie(String title, int year, int duration) {
}
```

- Incluido en Java 16.
- Muy esperado por la comunidad.
- Inmutables.
- Constructor canónico.

Records: Constructor Canónico

```java
public record Movie(String title, int year, int duration) {
    public Movie {
        if (title == null || title.isEmpty()) {
            throw new IllegalArgumentException();
        }
    }
}
```

- Se ejecuta **siempre**.

Records: Constructor Canónico

```
public record Movie(String title, List<String> cast) {
    public Movie {
        cast = List.copyOf(cast);
    }
}
```

- Copia defensiva para evitar modificar el contenido del record.

Sealed classes and interfaces

```
public sealed interface Shape {
    record Square(int side) implements Shape {}
    record Rectangle(int weight, int height) implements Shape {}
    record Circle(int radius) implements Shape {}
}
```

- Incluido en Java 17.
- Jerarquías de clases cerradas.
- `non-sealed`

non-sealed

```java
public sealed interface Shape {
    record Square(int side) implements Shape {}
    record Rectangle(int weight, int height) implements Shape {}
    record Circle(int radius) implements Shape {}
    non-sealed interface CustomShape extends Shape {}
}
```

Pattern matching for switch

```
var result = switch (obj) {
    case Integer i -> String.format("int %d", i);
    case Long l    -> String.format("long %d", l);
    case Double d  -> String.format("double %f", d);
    case String s  -> String.format("String %s", s);
    default        -> obj.toString();
};
```

- Incluido en Java 21.

Null patterns

```
var result = switch (obj) {
    case null      -> "null";
    case Integer i -> String.format("int %d", i);
    case Long l    -> String.format("long %d", l);
    case Double d  -> String.format("double %f", d);
    case String s  -> String.format("String %s", s);
    default        -> obj.toString();
};
```

Record patterns

```
var area = switch (this) {
    case Square(var side) -> side * side;
    case Rectangle(var weight, var height) -> weight * height;
    case Circle(var radius) -> Math.PI * Math.pow(radius, 2);
};
```

- Incluido en Java 21.
- Deconstructores, nos permite acceder a los componentes de los objectos.
- Exhaustiveness.

Guarded patterns

```
var result = switch (point) {
    case Point(var x, var y) when y == 0 -> processX(x);
    case Point(var x, var y) -> processXY(x, y);
};
```

- Podemos añadir condiciones adicionales usando `when`

Nested patterns

```
var result = switch (this) {
    case Square(Point(var x, var y), var side) -> ...;
    case Rectangle(Point(var x, var y), var weight, var height) -> ...;
    case Circle(Point(var x, var y), var radius) -> ...;
};
```

- Podemos anidar patrones

Unnamed variables and patterns

```
var result = switch (obj) {
    case Integer _ -> "int";
    case Long _    -> "long";
    case Double _  -> "double";
    case String _  -> "string";
    default        -> "other";
};
```

- Incluido en Java 22.
- Mejora para el pattern matching.
- Eliminar verbosidad.

Tipos de datos algebraicos

- AKA ADTs (algebraic data types).
- Viene de las matemáticas.
- Recursivo.
- Productos y sumas de tipos.
  `a + b` `a * b`
- Tienen propiedades algebraicas.

Otros lenguajes con soporte ADTs

- JVM:
  - Scala, Kotlin
- Rust
- C#
- TypeScript
- Otros lenguajes funcionales:
  - Haskell, F#, OCaml

ADTs

- ¿Cómo podemos representarlos en Java?
- Un `record` es un producto de tipos.
- Un `sealed interface` es una suma de tipos.
- ¿Pero eso para qué me sirve?
    - Estructuras de datos.
    - Estructuras de control.
    - DSLs
    - Manejo de errores.

ADTs: List

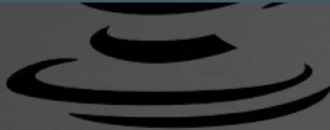
```java
sealed interface List<T> {
    record NonEmpty<T>(T head, List<T> tail) implements List<T> {}
    record Empty<T>() implements List<T> {}
}
```

ADTs: List (map)

```
sealed interface List<T> {
    default <R> List<R> map(Function<T, R> mapper) {
        return switch (this) {
            case NonEmpty<T>(var head, var tail)
                -> new NonEmpty<>(mapper.apply(head), tail.map(mapper));
            case Empty<T> _ -> new Empty<>();
        };
    }
}
```

ADTs: List (filter)

```
sealed interface List<T> {
    default List<T> filter(Predicate<T> filter) {
        return switch (this) {
            case NonEmpty<T>(var head, var tail) when filter.test(head)
                -> new NonEmpty<>(head, tail.filter(filter));
            case NonEmpty<T>(var head, var tail)
                -> tail.filter(filter);
            case Empty<T> _ -> new Empty<>();
        };
    }
}
```



ADTs: List (fold)

```
sealed interface List<T> {
    default T fold(T initial, BinaryOperator<T> operator) {
        return switch (this) {
            case NonEmpty<T>(var head, var tail)
                -> tail.fold(operator.apply(initial, head), operator);
            case Empty<T> _ -> initial;
        };
    }
}
```

ADTs: Tree

```
sealed interface Tree<T> {
    record Node<T>(T value, Tree<T> left, Tree<T> right) implements Tree<T> { }
    record Leaf<T>(T value) implements Tree<T> {}
}
```

ADTs: Optional

```
sealed interface Optional<T> {
    record Empty<T>() implements Optional<T> { }
    record Present<T>(T value) implements Optional<T> {}
}
```

ADTs: Optional (map)

```java
sealed interface Optional<T> {
    default <R> Optional<R> map(Function<T, R> mapper) {
        return switch (this) {
            case Present<T>(var value) -> new Present<>(mapper.apply(value));
            case Empty<T> _ -> new Empty<>();
        };
    }
}
```

ADTs: Optional (filter)

```java
sealed interface Optional<T> {
    default Optional<T> filter(Predicate<T> filter) {
        return switch (this) {
            case Present<T>(var value) when filter.test(value) -> this;
            case Present<T> _ -> new Empty<>();
            case Empty<T> _ -> new Empty<>();
        };
    }
}
```

ADTs: Optional (fold)

```
sealed interface Optional<T> {
    default <R> R fold(Supplier<R> onEmpty, Function<T, R> onPresent) {
        return switch (this) {
            case Present<T>(var value) -> onPresent.apply(value);
            case Empty<T> _ -> onEmpty.get();
        };
    }
}
```

ADTs: Either

```
sealed interface Either<L, R> {
    record Left<L, R>(L left) implements Either<L, R> { }
    record Right<L, R>(R right) implements Either<L, R> {}
}
```

ADTs: DSLs

```
sealed interface Expr {
    record Val(int value) implements Expr {}
    record Sum(Expr left, Expr right) implements Expr {}
    record Diff(Expr left, Expr right) implements Expr {}
    record Times(Expr left, Expr right) implements Expr {}
}
```

ADTs: DSLs

```
sealed interface Expr {
    default int evaluate() {
        return switch (this) {
            case Val(var value) -> value;
            case Sum(var left, var right) -> left.evaluate() + right.evaluate();
            case Diff(var left, var right) -> left.evaluate() - right.evaluate();
            case Times(var left, var right) -> left.evaluate() * right.evaluate();
        };
    }
}
```

ADTs: DSLs

```
sealed interface Expr {
    default String asString() {
        return switch (this) {
            case Val(var value) -> String.valueOf(value);
            case Sum(var left, var right) -> "(" + left.asString() + "+" + right.asString() + ")";
            case Diff(var left, var right) -> "(" + left.asString() + "-" + right.asString() + ")";
            case Times(var left, var right) -> "(" + left.asString() + "*" + right.asString() + ")";
        };
    }
}
```

ADTs: DSLs

```
sealed interface Expr {
    default void print() {
        System.out.println(asString() + "=" + evaluate());
    }
}
```

ADTs: DSLs

```java
static void main() {
    sum(val(1), val(2)).print();
    times(diff(val(10), val(8)), val(2)).print();
}
```

```
(1+2)=3
((10-8)*2)=4
```

ADTs: Json

```java
sealed interface Json {
    enum JsonNull implements Json { NULL }
    enum JsonBoolean implements Json { TRUE, FALSE }
    record JsonString(String value) implements Json {}
    record JsonNumber(Number value) implements Json {}
    record JsonObject(Map<String, Json> value) implements Json {}
    record JsonArray(List<Json> value) implements Json {}
}
```

ADTs: Json

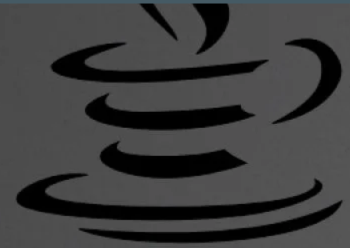
```
sealed interface Json {
    default String asString() {
        return switch (this) {
            case JsonNull _ -> "null";
            case JsonBoolean b -> switch (b) {
                case TRUE -> "true";
                case FALSE -> "false";
            };
            ...
        };
    }
}
```

ADTs: Json

```
sealed interface Json {
    default String asString() {
        return switch (this) {
            ...
            case JsonString(var value) -> "\"" + value + "\"";
            case JsonNumber(var value) -> String.valueOf(value);
            ...
        };
    }
}
```

ADTs: Json

```
sealed interface Json {
    default String asString() {
        return switch (this) {
            ...
            case JsonObject(var map)
                -> map.entrySet().stream()
                    .map(entry ->  "\"" + entry.getKey() + "\":" + entry.getValue().asString())
                    .collect(joining(",", "{", "}"));
            ...
        };
    }
}
```



ADTs: Json

```
sealed interface Json {
    default String asString() {
        return switch (this) {
            ...
            case JsonArray(var array)
                -> array.stream()
                    .map(Json::asString)
                    .collect(joining(",", "[", "]"));
        };
    }
}
```

ADTs: Json

```
static void main() {
    var json = array(
        object(
            entry("name", string("Toni")),
            entry("age", number(46)),
            entry("old", JsonBoolean.TRUE)),
        object(
            entry("name", string("Baby")),
            entry("age", JsonNull.NULL),
            entry("old", JsonBoolean.FALSE))
    );
    System.out.println(json.asString());
}
```

ADTs: Json

```
[{"old":true,"name":"Toni","age":46},{"old":false,"name":"Baby","age":null}]
```

ADTs: Errores

```
interface MovieRepository {
    MovieResponse create(Movie movie);
}

sealed interface MovieResponse permits MovieCreated, MovieError {}

record MovieCreated(UUID id) implements MovieResponse {}
```

ADTs: Errores

```java
sealed interface MovieError extends MovieResponse {
    record DuplicatedMovie(UUID id) implements MovieError {}
    record InvalidDuration(int duration) implements MovieError {}
    record InvalidYear(int year) implements MovieError {}
    record InvalidStars(int stats) implements MovieError {}
    record EmptyTitle() implements MovieError {}
    record EmptyDirector() implements MovieError {}
    record EmptyCast() implements MovieError {}
    record DuplicatedActor(String actor) implements MovieError {}
}
```

- Definir errores de dominio.

ADTs: Errores

```
@PostMapping("/movies")
public ResponseEntity<UUID> create(@RequestBody Movie movie) {
    var result = repository.create(movie);
    return switch (result) {
        case MovieCreated(var id) -> ResponseEntity.ok(id);
        case MovieError e -> ResponseEntity.of(toProblem(e)).build();
    };
}
```

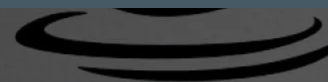
ADTs: Errores

```java
static ProblemDetail toProblem(CreateMovieResponse.MovieError error) {
    var detail = switch (error) {
      case DuplicatedMovie(int id) -> "duplicated movie with id: " + id;
      case InvalidDuration(var duration) -> "invalid duration: " + duration;
      case InvalidYear(var duration) -> "invalid year: " + duration;
      case InvalidStars(var stars) -> "invalid stars: " + stars;
      case EmptyTitle() -> "title cannot be empty";
      case EmptyDirector() -> "director cannot be empty";
      case EmptyCast() -> "cast cannot be empty";
      case DuplicatedActor(var actor) -> "duplicated actor: " + actor;
    };
    return ProblemDetail.forStatusAndDetail(BAD_REQUEST, detail);
}
```

ADTs: Either

```java
interface MovieRepository {
    Either<MovieError, UUID> create(Movie movie);
}
```

```java
@PostMapping("/movies")
public ResponseEntity<UUID> create(@RequestBody Movie movie) {
    var result = repository.create(movie);
    return result.fold(
        error -> ResponseEntity.of(toProblem(error)).build(),
        ResponseEntity::ok);
}
```



Próximamente ⌚

Primitive types in patterns

```
var result = switch (obj) {
    case int i      -> String.format("int %d", i);
    case long l     -> String.format("long %d", l);
    case double d   -> String.format("double %f", d);
    case String s   -> String.format("String %s", s);
    default         -> obj.toString();
};
```

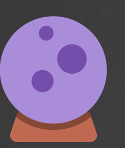
- Preview en Java 23 (sep 2024).

Primitive types in patterns

```
jshell> int i = 0;
i ==> 0
jshell> i instanceof byte b
$2 ==> true
```

```
jshell> int i = 123213;
i ==> 123213
jshell> i instanceof byte b
$4 ==> false
```

- Castings seguros

Bola de cristal 

Derived Record Creation

```
Point newPoint = oldPoint with {
    x *= 2;
    y *= 2;
};
```

- Draft.
- AKA withers.

Static patterns

```
var result = switch (optional) {
    case Optional.of(var value) -> value.toString();
    case Optional.empty() -> "empty";
};
```

- Early work.
- AKA deconstructors.
- Mejora para pattern matching.
- Cualquier clase.

Constant patterns

```
var result = switch (optional) {
    case Point(0, var y) -> process(y);
    case Point(var x, var y) -> process(x, y);
};
```

¿Qué falta todavía? 🤓

- Tail recursion.
- Soporte de tipos de datos primitivos en genéricos.

¿Preguntas?

¡Gracias! 💖

JEPs

- Switch expressions 14
- Records 16
- Sealed Classes and interfaces 17
- Pattern matching for switch 21
- Record patterns 21
- Unnamed variables and patterns 22
- Primitive types in patterns 23 (1st preview)
- Derived record creation Candidate

Documentación Oficial

- Switch Expressions
- Record Classes
- Sealed Classes
- Pattern Matching
- Unnamed variables and patterns

Artículos / Videos

- Data Oriented Programming Brian Goetz
- Data Oriented Programming with Java 21 Nicolai Parlog
- Java 23: Restoring the Balance with Primitive Patterns Nicolai Parlog
- Java Language Update 2023 Brian Goetz
- Why ADTs are important? Bartosz Milewski

Enlaces

- Slides HTML: https://tonivade.es/commitconf24/slides.html
- Slides PDF: https://tonivade.es/commitconf24/slides.pdf