Pattern Matching

From Small Enhancement to Major Feature

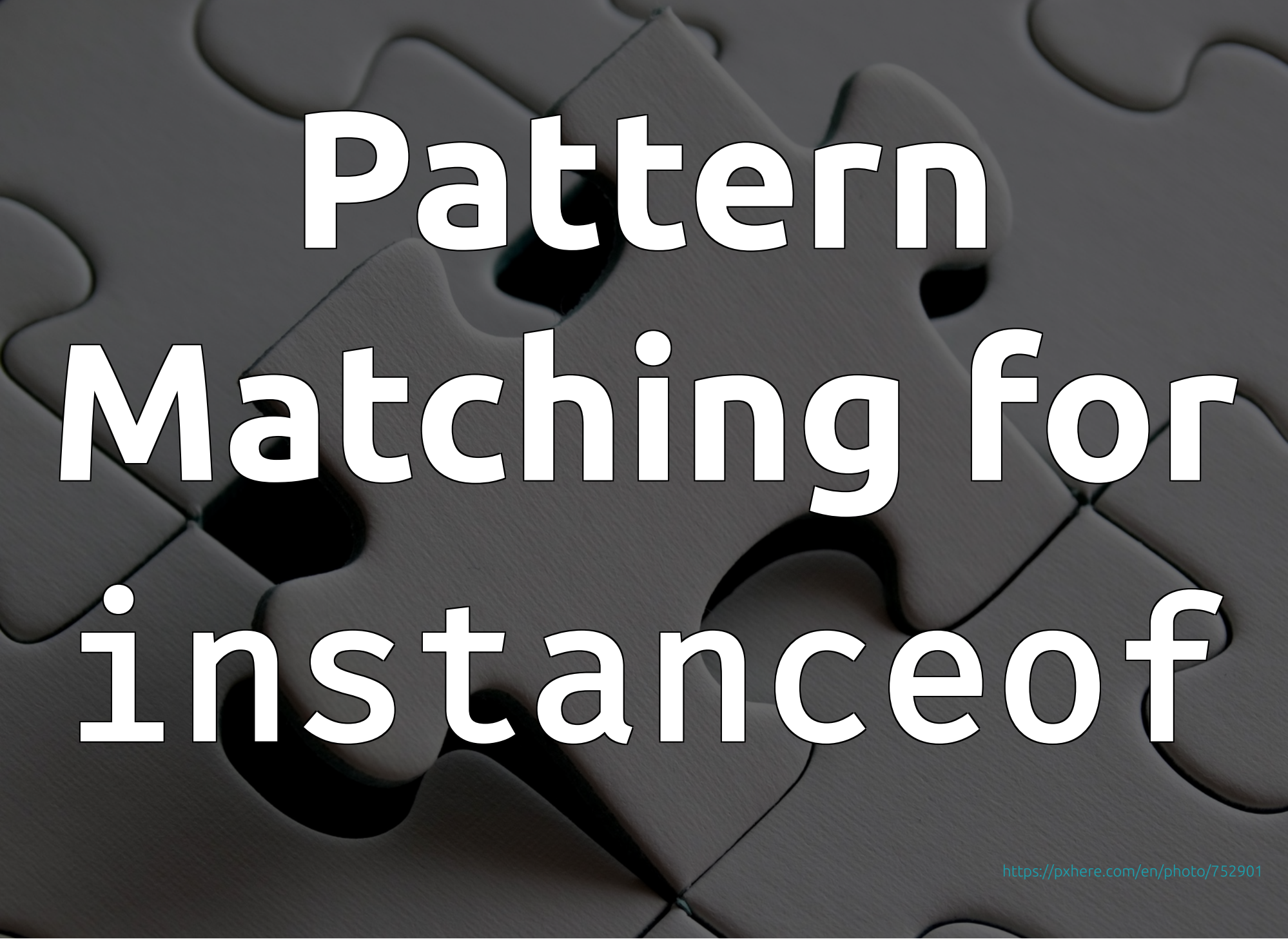
Hanno Embregts

@hannotify

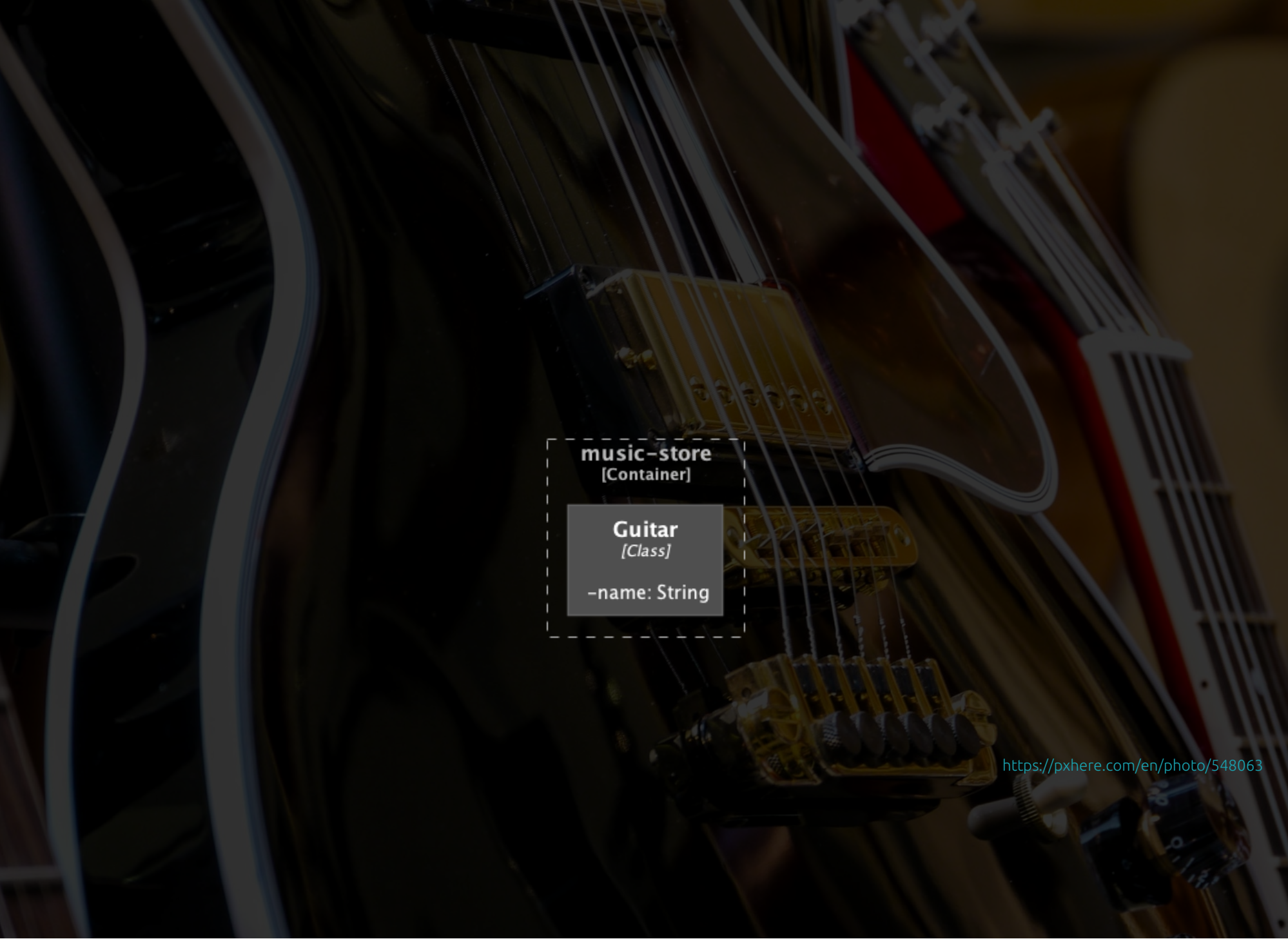

DEVOXX
BELGIUM


Small Enhancement
https://gph.is/g/ZPJNoPQ


Major Feature
https://thumbs.gfycat.com/DefiantElasticGadwall.webp



https://pxhere.com/en/photo/752901



https://pxhere.com/en/photo/548063

Instanceof-and-cast

```
1 if (product instanceof Guitar) {
2     Guitar lesPaul =
3                     (Guitar) product;
4     // use lesPaul
5 }
```

Instanceof-and-cast

```
1 if (product instanceof Guitar) { // 1. is product a Guitar?
2     Guitar lesPaul =
3                     (Guitar) product;
4     // use lesPaul
5 }
```

Instanceof-and-cast

```
1 if (product instanceof Guitar) { // 1. is product a Guitar?
2     Guitar lesPaul =
3                     (Guitar) product; // 2. perform conversion
4     // use lesPaul
5 }
```

Instanceof-and-cast

```
1 if (product instanceof Guitar) { // 1. is product a Guitar?
2     Guitar lesPaul = // 3. declare variable, bind value
3                      (Guitar) product; // 2. perform conversion
4     // use lesPaul
5 }
```

Improve the situation

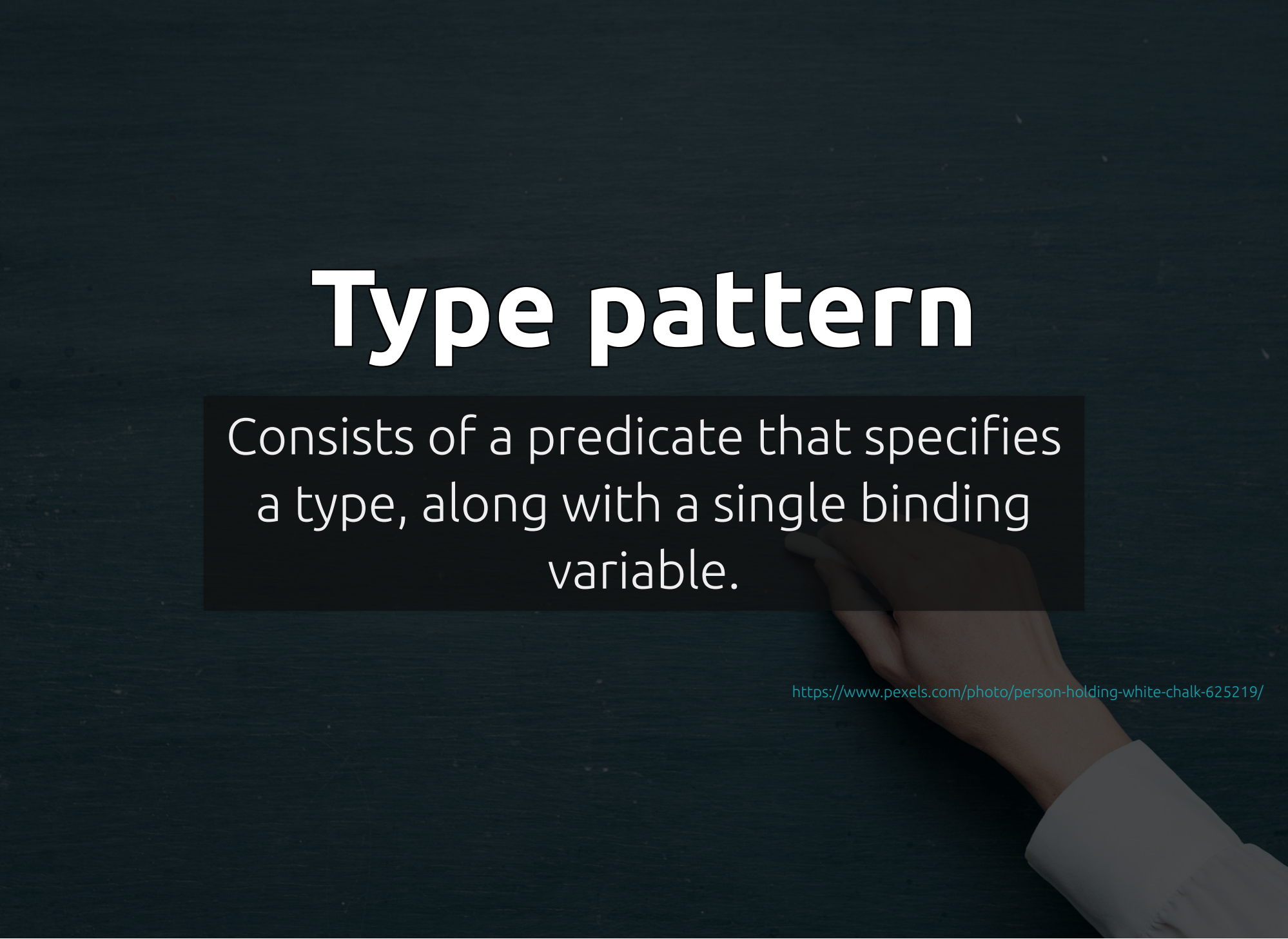
```
1 if (product instanceof Guitar) { // 1. is product a Guitar?
2     Guitar lesPaul = // 3. declare variable, bind value
3                      (Guitar) product; // 2. perform conversion
4     // use lesPaul
5 }
```

Improve the situation

```
1 if (product instanceof Guitar lesPaul) {
2     // use lesPaul
3 }
```

Type pattern

Consists of a predicate that specifies a type, along with a single binding variable.



https://www.pexels.com/photo/person-holding-white-chalk-625219/

Pattern matching

Allows the conditional extraction of components from objects to be expressed more concisely and safely.

https://www.pexels.com/photo/person-holding-white-chalk-625219/

Declaring 'in the middle'

```
1 if (product instanceof Guitar lesPaul) {
2     // use lesPaul
3 }
```

Scoping

'Regular' local variable ('block scoping')

- The block in which it is declared.

```
1 void playTunedGuitar() {
2     Guitar lesPaul = new Guitar("Les Paul");
3 
4     if (!lesPaul.isInTune()) {
5         Guitar fenderStrat = new Guitar("Fender Stratocaster");
6         fenderStrat.play();
7     }
8 }
```

Scoping

'Regular' local variable ('block scoping')

- The block in which it is declared.

```
 1 void playTunedGuitar() {
 2     Guitar lesPaul = new Guitar("Les Paul");
 3
 4     if (!lesPaul.isInTune()) {
 5         Guitar fenderStrat = new Guitar("Fender Stratocaster");
 6         fenderStrat.play();
 7         // fenderStrat is in scope
 8     }
 9     // fenderStrat is not in scope
10 }
```

Scoping

Pattern binding variable ('flow scoping')

- The set of places where it would definitely be assigned.

```
1 if (product instanceof Guitar lesPaul) {
2     // can use lesPaul here
3 } else {
4     // can't use lesPaul here
5 }
```

Scoping

Pattern binding variable ('flow scoping')

- The set of places where it would definitely be assigned.

```
1 if (product instanceof Guitar lesPaul && lesPaul.isInTune()) {
2     // can use lesPaul here
3 } else {
4     // can't use lesPaul here
5 }
```

Scoping

Pattern binding variable ('flow scoping')

- The set of places where it would definitely be assigned.

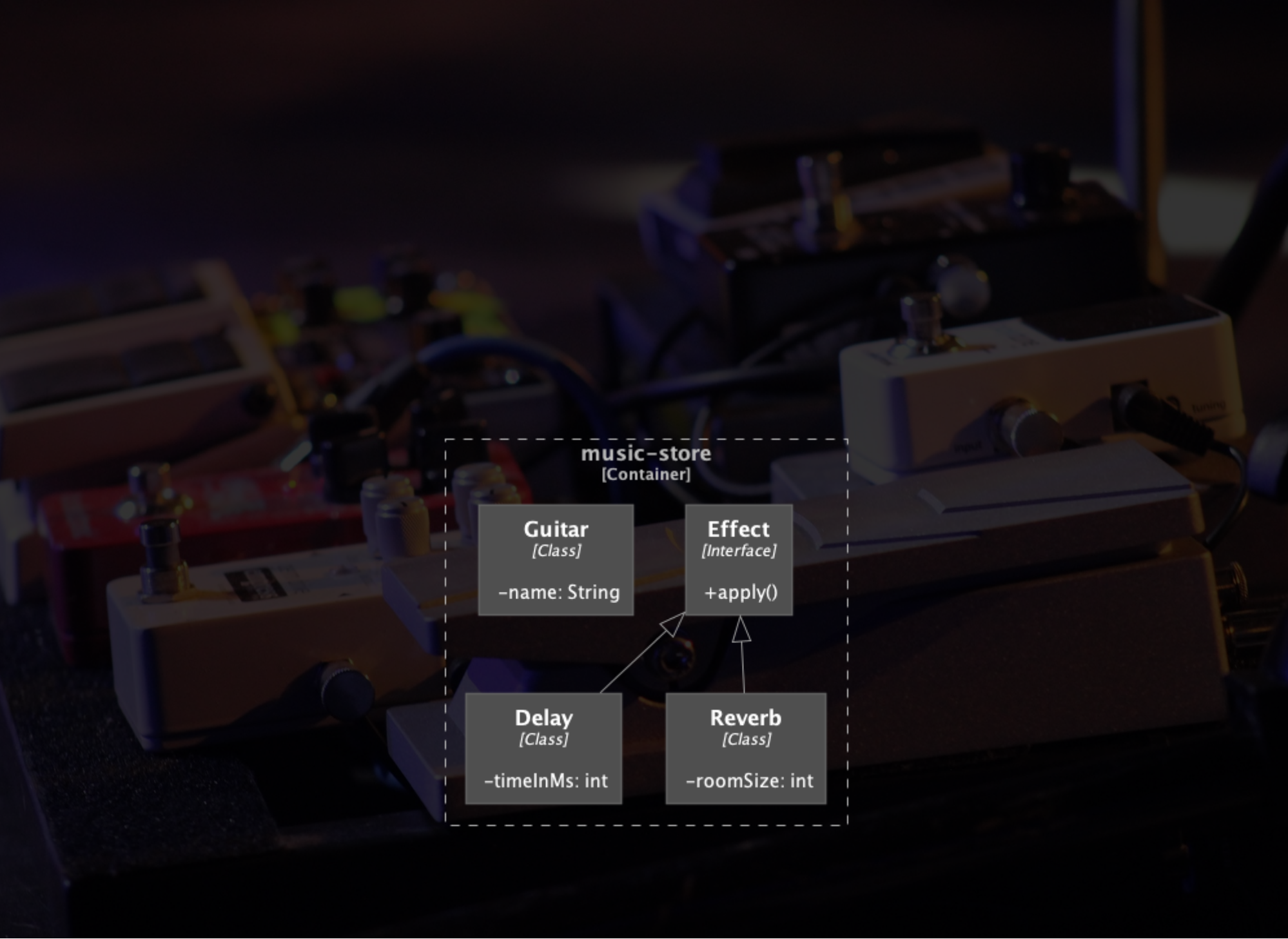
```
1 boolean isTunedGuitar(Object product) {
2     if (!(product instanceof Guitar lesPaul)) {
3         return false;
4     }
5
6     return lesPaul.isInTune();
7 }
```

Scoping

Pattern binding variable ('flow scoping')

- The set of places where it would definitely be assigned.

```
1 boolean isTunedGuitar(Object product) {
2     if (!(product instanceof Guitar lesPaul)) {
3         return false;
4     }
5
6     // This code is only reachable if 'product' is
7     // a Guitar, so 'lesPaul' is in scope.
8     return lesPaul.isInTune();
9 }
```


music-store
[Container]
Guitar
[Class]
-name: String
Effect
[Interface]
+apply()
Delay
[Class]
-timeInMs: int
Reverb
[Class]
-roomSize: int

Scoping

Pattern binding variable ('flow scoping')

- The set of places where it would definitely be assigned.

```
1 void test(Effect effect) {
2     if (effect instanceof Reverb stockEffect)
3         stockEffect.setRoomSize(25);
4     else if (effect instanceof Delay stockEffect)
5         stockEffect.setTimeInMs(200);
6 }
```


Overdrive Channel
Digital FX
TREBLE
VOLUME
PRESET/
ADJUST
FX LEVEL

Demo

- Simplify implementation of `equals`
- Loop through a set of `Effects` and apply 'pattern matching for instanceof'

https://pxhere.com/en/photo/1458897

Benefits

- Nearly 100% of casts will just disappear!
- More concise
- Eliminates cut/paste errors

`instanceof` grammar

The `instanceof` grammar is extended accordingly:

```
1 RelationalExpression:
2   ...
3   RelationalExpression instanceof ReferenceType
4   RelationalExpression instanceof Pattern
5
6 Pattern:
7   ReferenceType Identifier
```

https://openjdk.java.net/jeps/305

It's a kind of Pattern

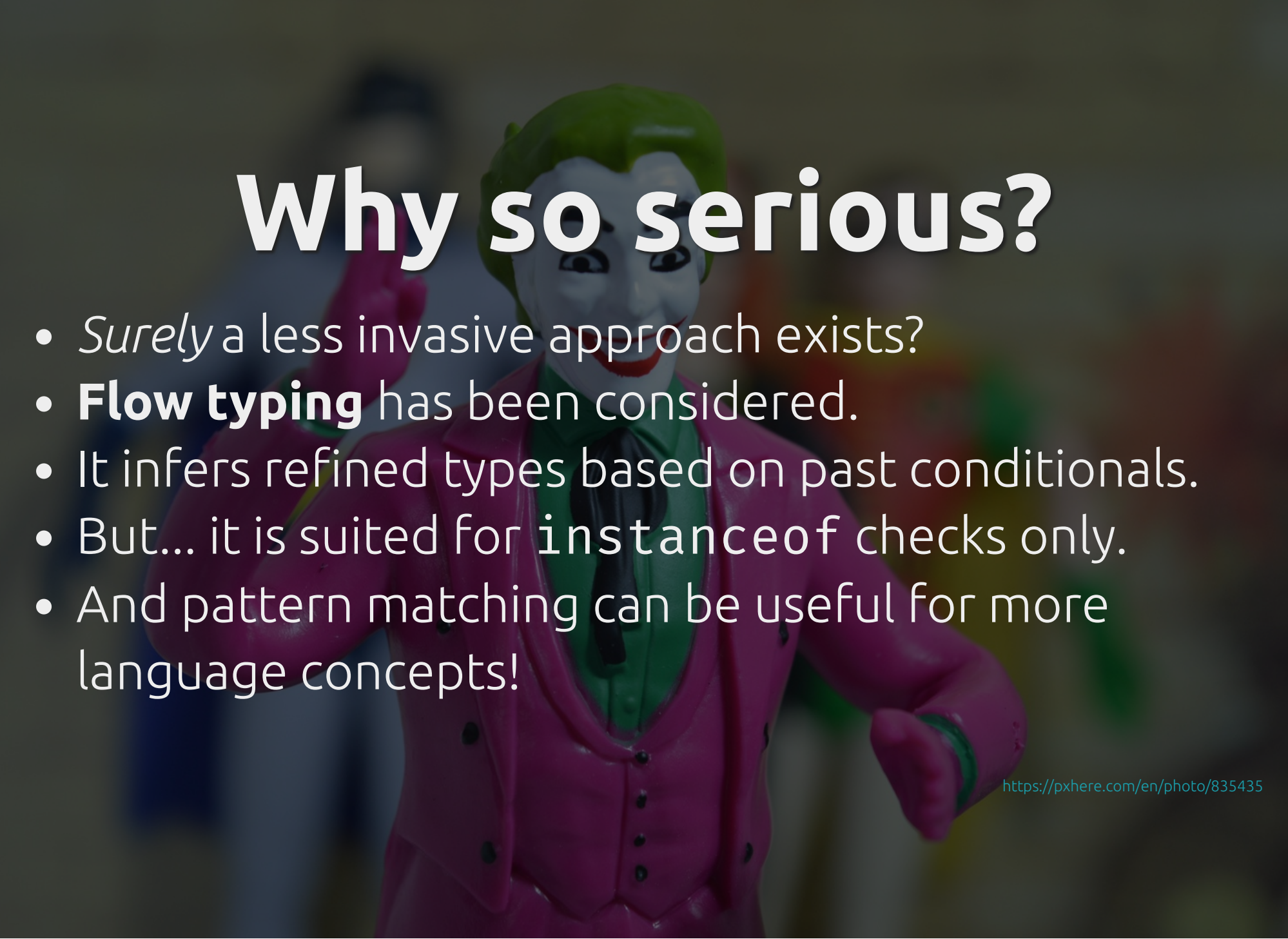
type pattern

```
Guitar lesPaul
```

https://www.pexels.com/photo/gray-metal-statue-of-man-raising-hand-near-dock-825430/

Feature Status

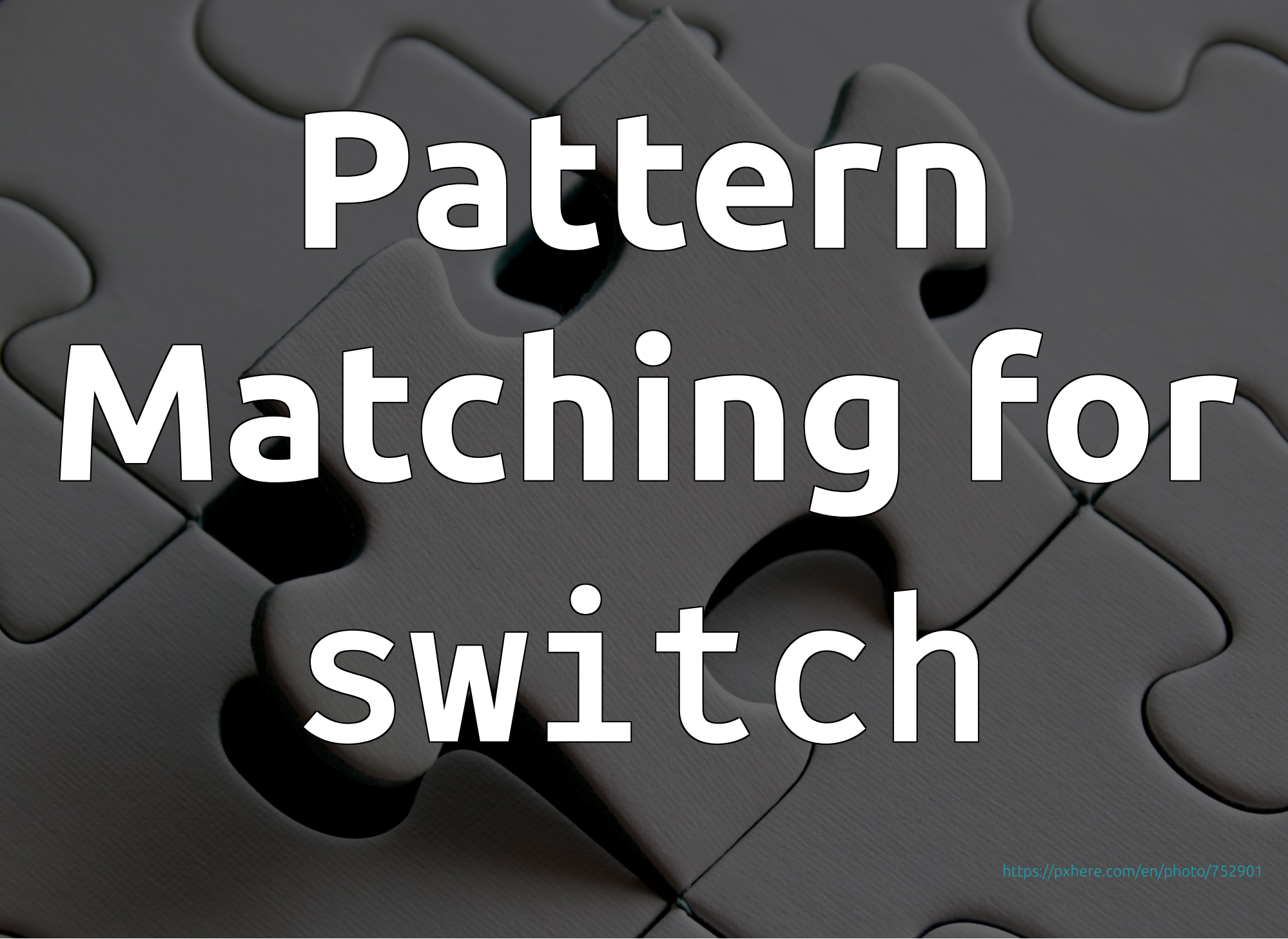
Java version	Feature status	JEP
14	Preview	JEP 305
15	Second preview	JEP 375
16	Final	JEP 394



Why so serious?

- *Surely* a less invasive approach exists?
- **Flow typing** has been considered.
- It infers refined types based on past conditionals.
- But... it is suited for `instanceof` checks only.
- And pattern matching can be useful for more language concepts!

https://pxhere.com/en/photo/835435


Pattern
Matching for
switch
https://pxhere.com/en/photo/752901

Disclaimer

I can't tell you when the following features are coming to Java.
Also: syntax and implementation specifics may still change.

```
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay) {
        Delay de = (Delay) effect;
        formatted = String.format("Delay active of %d ms.", de.getTim
    } else if (effect instanceof Reverb) {
        Reverb re = (Reverb) effect;
        formatted = String.format("Reverb active of type %s and roomS
    } else if (effect instanceof Overdrive) {
        Overdrive ov = (Overdrive) effect;
        formatted = String.format("Overdrive active with gain %d.", o
    } else if (effect instanceof Tremolo) {
        Tremolo tr = (Tremolo) effect;
        formatted = String.format("Tremolo active with depth %d and r
    } else if (effect instanceof Tuner) {
        Tuner tu = (Tuner) effect;
```

```
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay) {
        Delay de = (Delay) effect;
        formatted = String.format("Delay active of %d ms.", de.getTim
    } else if (effect instanceof Reverb) {
        Reverb re = (Reverb) effect;
        formatted = String.format("Reverb active of type %s and roomS
    } else if (effect instanceof Overdrive) {
        Overdrive ov = (Overdrive) effect;
        formatted = String.format("Overdrive active with gain %d.", o
    } else if (effect instanceof Tremolo) {
        Tremolo tr = (Tremolo) effect;
        formatted = String.format("Tremolo active with depth %d and r
    } else if (effect instanceof Tuner) {
        Tuner tu = (Tuner) effect;
```

```
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay de) {
        formatted = String.format("Delay active of %d ms.", de.getTim
    } else if (effect instanceof Reverb re) {
        formatted = String.format("Reverb active of type %s and roomS
    } else if (effect instanceof Overdrive ov) {
        formatted = String.format("Overdrive active with gain %d.", o
    } else if (effect instanceof Tremolo tr) {
        formatted = String.format("Tremolo active with depth %d and r
    } else if (effect instanceof Tuner tu) {
        formatted = String.format("Tuner active with pitch %d. Muting
    } else if (effect instanceof EffectLoop el) {
        formatted = el.getEffects().stream().map(this::apply).collect
    } else {
        formatted = String.format("Unknown effect active: %s.", effec
```

```
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay de) {
        formatted = String.format("Delay active of %d ms.", de.getTim
    } else if (effect instanceof Reverb re) {
        formatted = String.format("Reverb active of type %s and roomS
    } else if (effect instanceof Overdrive ov) {
        formatted = String.format("Overdrive active with gain %d.", o
    } else if (effect instanceof Tremolo tr) {
        formatted = String.format("Tremolo active with depth %d and r
    } else if (effect instanceof Tuner tu) {
        formatted = String.format("Tuner active with pitch %d. Muting
    } else if (effect instanceof EffectLoop el) {
        formatted = el.getEffects().stream().map(this::apply).collect
    } else {
        formatted = String.format("Unknown effect active: %s.", effec
```

Switch expression

```
1 String apply(Effect effect) {
2     return switch(effect) {
3         default             -> String.format("Unknown effect active: %s
4     };
5 }
```

Switch expression

```
1 String apply(Effect effect) {
2     return switch(effect) {
3         case Delay de      -> String.format("Delay active of %d ms.",
4         case Reverb re     -> String.format("Reverb active of type %s
5         default            -> String.format("Unknown effect active: %s
6     };
7 }
```

Switch expression

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de       -> String.format("Delay active of %d ms.",
4          case Reverb re      -> String.format("Reverb active of type %s
5          case Overdrive ov   -> String.format("Overdrive active with ga
6          case Tremolo tr     -> String.format("Tremolo active with dept
7          case Tuner tu       -> String.format("Tuner active with pitch
8          default             -> String.format("Unknown effect active: %
9      };
10 }
```

Switch expression

```
String apply(Effect effect) {
    return switch(effect) {
        case Delay de       -> String.format("Delay active of %d ms.",
        case Reverb re      -> String.format("Reverb active of type %s
        case Overdrive ov   -> String.format("Overdrive active with ga
        case Tremolo tr     -> String.format("Tremolo active with dept
        case Tuner tu       -> String.format("Tuner active with pitch
        case EffectLoop el -> el.getEffects().stream().map(this::appl
        default             -> String.format("Unknown effect active: %
    };
}
```

Switch expression

```
 1 String apply(Effect effect) {
 2     return switch(effect) {
 3         case Delay de       -> String.format("Delay active of %d ms.",
 4         case Reverb re      -> String.format("Reverb active of type %s
 5         case Overdrive ov   -> String.format("Overdrive active with ga
 6         case Tremolo tr     -> String.format("Tremolo active with dept
 7         case Tuner tu       -> String.format("Tuner active with pitch
 8         case EffectLoop el -> el.getEffects().stream().map(this::appl
 9         default             -> String.format("Unknown effect active: %
10     };
11 }
```

Benefits

- A single expression instead of many assignments
- Less error-prone (in adding cases)
- More concise
- Safer - the compiler can check for missing cases

switch grammar

```
SwitchBlock:
  { SwitchRule {SwitchRule} }
  { {SwitchBlockStatementGroup} {SwitchLabel :} }

SwitchRule:
  SwitchLabel -> Expression ;
  SwitchLabel -> Block
  SwitchLabel -> ThrowStatement

SwitchBlockStatementGroup:
  SwitchLabel : {SwitchLabel :} BlockStatements

SwitchLabel:
  case CaseConstant {, CaseConstant}
  default
```

https://docs.oracle.com/javase/specs/jls/se15/html/jls-14.html#jls-14.11

switch grammar

```
SwitchBlock:
  { SwitchRule {SwitchRule} }
  { {SwitchBlockStatementGroup} {SwitchLabel :} }

SwitchRule:
  SwitchLabel -> Expression ;
  SwitchLabel -> Block
  SwitchLabel -> ThrowStatement

SwitchBlockStatementGroup:
  SwitchLabel : {SwitchLabel :} BlockStatements

SwitchLabel:
  case CaseConstant {, CaseConstant}
  default
```

https://docs.oracle.com/javase/specs/jls/se15/html/jls-14.html#jls-14.11

switch grammar

```
SwitchBlock:
  { SwitchRule {SwitchRule} }
  { {SwitchBlockStatementGroup} {SwitchLabel :} }

SwitchRule:
  SwitchLabel -> Expression ;
  SwitchLabel -> Block
  SwitchLabel -> ThrowStatement

SwitchBlockStatementGroup:
  SwitchLabel : {SwitchLabel :} BlockStatements

SwitchLabel:
  case Pattern {, Pattern}
  default
```


It's a kind of Pattern
constant pattern
GuitarType.TELECASTER
https://www.pexels.com/photo/gray-metal-statue-of-man-raising-hand-near-dock-825430/

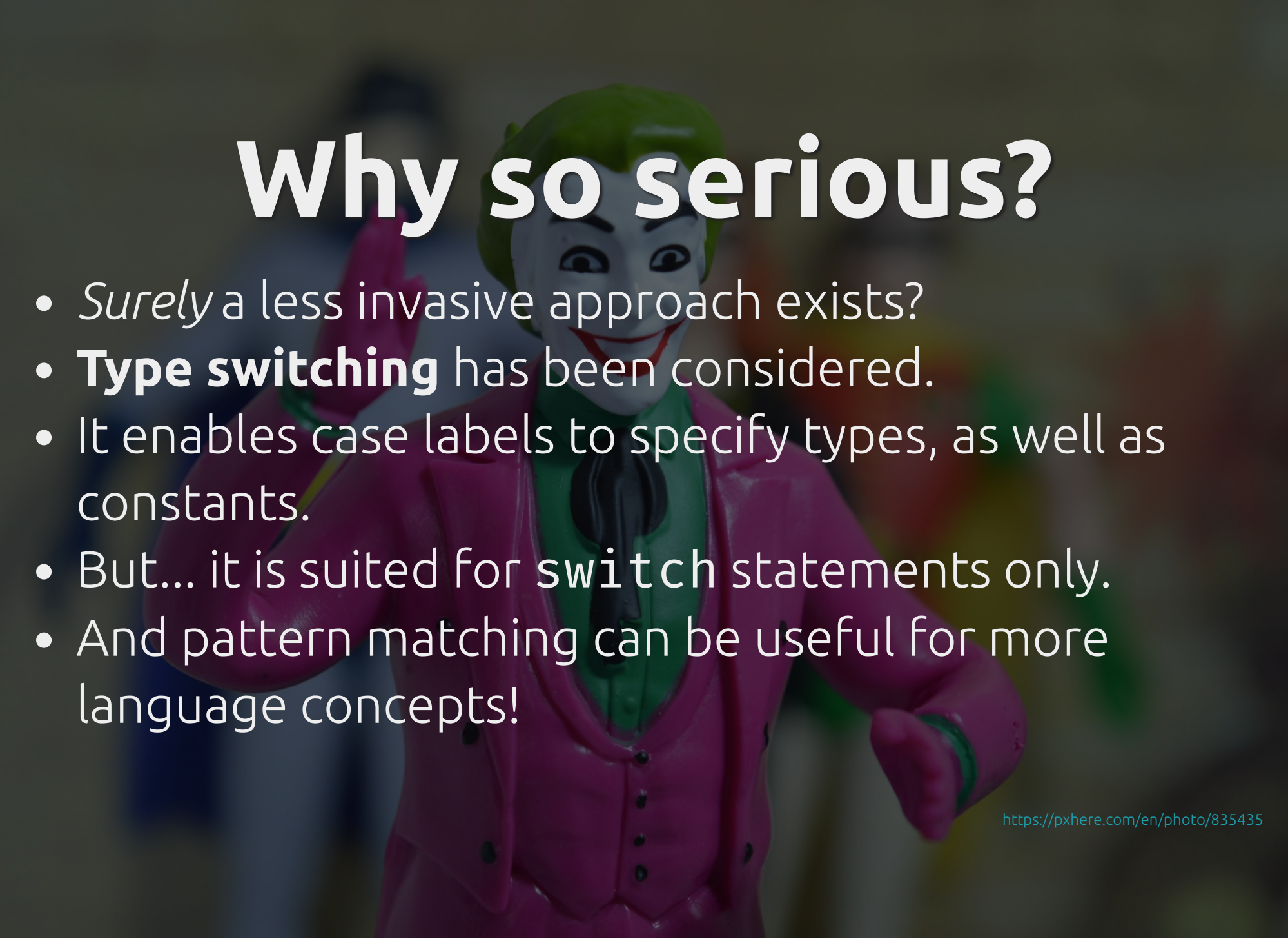
Feature Status

Java version	Feature status	JEP
n/a	Draft	JEP draft 8213076

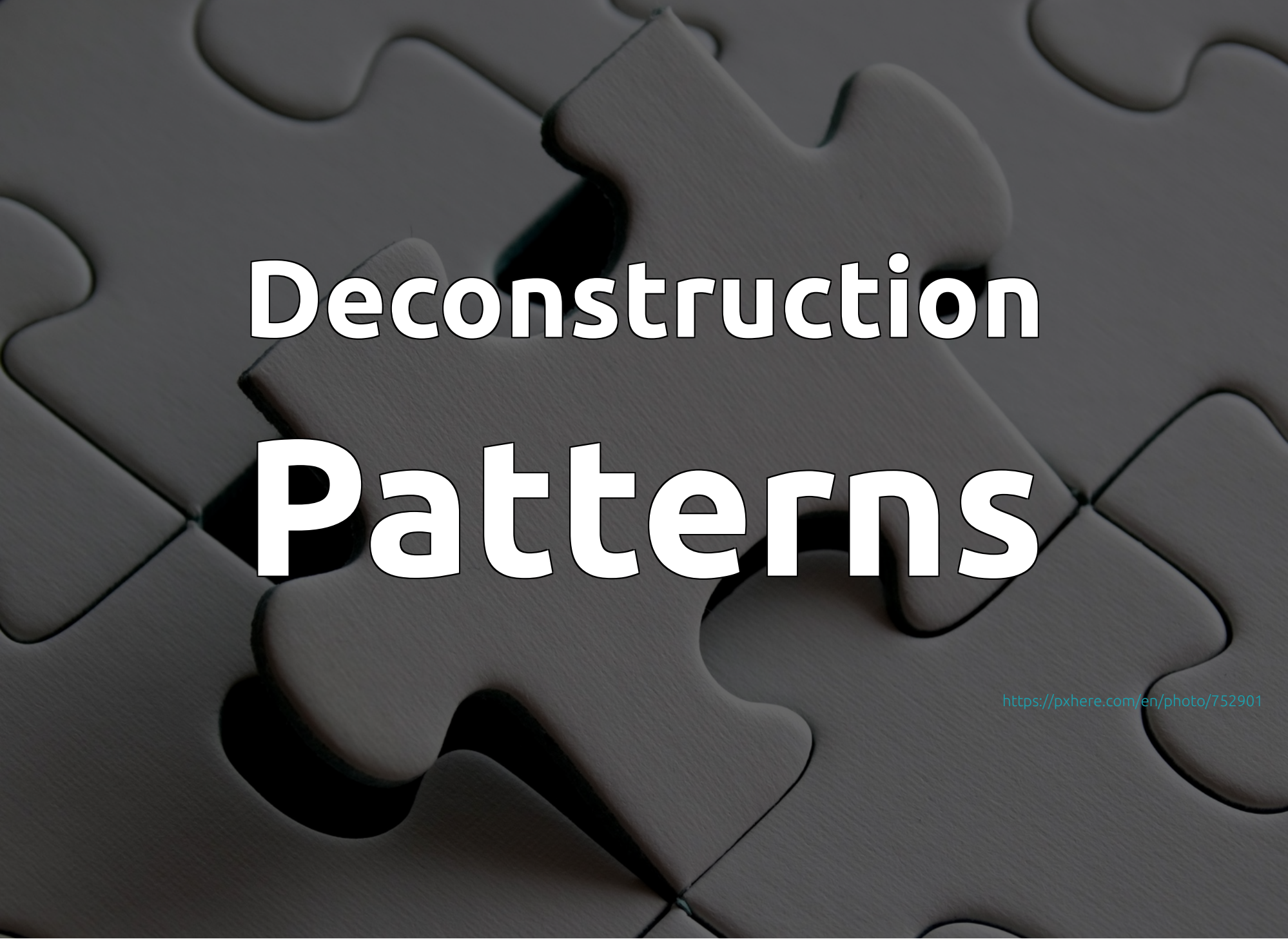
https://openjdk.java.net/jeps/8213076

Why so serious?

- *Surely* a less invasive approach exists?
- **Type switching** has been considered.
- It enables case labels to specify types, as well as constants.
- But... it is suited for `switch` statements only.
- And pattern matching can be useful for more language concepts!



https://pxhere.com/en/photo/835435


Deconstruction
Patterns
https://pxhere.com/en/photo/752901

Here be dragons!

We can't be sure **at all** that the following features will appear in Java as depicted. They can change a **lot** in the meantime.

https://www.pexels.com/photo/dragon-festival-during-nighttime-6068535/

Deconstruction patterns

```
 1 String apply(Effect effect) {
 2     return switch(effect) {
 3         case Delay de       -> String.format("Delay active of %d ms.",
 4         case Reverb re      -> String.format("Reverb active of type %s
 5         case Overdrive ov   -> String.format("Overdrive active with ga
 6         case Tremolo tr     -> String.format("Tremolo active with dept
 7         case Tuner tu       -> String.format("Tuner active with pitch
 8         case EffectLoop el -> el.getEffects().stream().map(this::appl
 9         default             -> String.format("Unknown effect active: %
10     };
11 }
```

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de       -> String.format("Delay active of %d ms.",
4          case Reverb re      -> String.format("Reverb active of type %s
5          case Overdrive(int gain) -> String.format("Overdrive active w
6          case Tremolo tr     -> String.format("Tremolo active with dept
7          case Tuner tu       -> String.format("Tuner active with pitch
8          case EffectLoop el -> el.getEffects().stream().map(this::appl
9          default             -> String.format("Unknown effect active: %
10     };
11 }
```

Pattern definition

```
1 public class Overdrive implements Effect {
2     private final int gain;
3
4     public Overdrive(int gain) {
5         this.gain = gain;
6     }
7 }
```

Pattern definition

```
 1 public class Overdrive implements Effect {
 2     private final int gain;
 3
 4     public Overdrive(int gain) {
 5         this.gain = gain;
 6     }
 7
 8     public pattern Overdrive(int gain) {
 9         gain = this.gain;
10     }
11 }
```

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de       -> String.format("Delay active of %d ms.",
4          case Reverb re      -> String.format("Reverb active of type %s
5          case Overdrive(int gain) -> String.format("Overdrive active w
6          case Tremolo tr     -> String.format("Tremolo active with dept
7          case Tuner tu       -> String.format("Tuner active with pitch
8          case EffectLoop el -> el.getEffects().stream().map(this::appl
9          default             -> String.format("Unknown effect active: %
10     };
11 }
```

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d
4          case Reverb(String name, int roomSize) -> String.format("Reve
5          case Overdrive(int gain) -> String.format("Overdrive active w
6          case Tremolo(int depth, int rate) -> String.format("Tremolo a
7          case Tuner(int pitchInHz) -> String.format("Tuner active with
8          case EffectLoop(Set<Effect> effects) -> effects.stream().map(
9          default -> String.format("Unknown effect active: %s.", effect
10     };
11 }
```

Pattern composition

```
1 static boolean containsReverbAndDelayWithEqualProperties(EffectLoop ef
2
3 }
```

Pattern composition

```
1  static boolean containsReverbAndDelayWithEqualProperties(EffectLoop e
2      return effectLoop.getEffects().stream()
3          .filter(e -> e instanceof Delay || e instanceof Reverb)
4          .map(dr -> {
5              if (dr instanceof Delay d) {
6                  return d.getTimeInMs();
7              } else {
8                  Reverb r = (Reverb) dr;
9                  return r.getRoomSize();
10             }
11
12         }).distinct().count() == 1;
13 }
```

Pattern composition

```
1 static boolean containsReverbAndDelayWithEqualProperties(EffectLoop ef
2     if (effectLoop instanceof EffectLoop(Delay(int timeInMs), Reverb(S
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```

Var and any patterns

```
1 // Pre-Java 10
2 Guitar telecaster = new Guitar("Fender Telecaster Baritone Blacktop",
3
4 // Java 10
5 var telecaster = new Guitar("Fender Telecaster Baritone Blacktop", Gui
```

https://openjdk.java.net/jeps/286

Var and any patterns

```
1 static boolean containsReverbAndDelayWithEqualProperties(EffectLoop ef
2     if (effectLoop instanceof EffectLoop(Delay(int timeInMs), Reverb(S
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```

Var and any patterns

```
1 static boolean containsReverbAndDelayWithEqualProperties(EffectLoop ef
2     if (effectLoop instanceof EffectLoop(Delay(var timeInMs), Reverb(v
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```


TO START
PRESS ANY KEY
http://gph.is/2lFlHIK
"To start, press any key."
Where's the "any" key?
90s90s90s

Var and any patterns

```
1 static boolean containsReverbAndDelayWithEqualProperties(EffectLoop ef
2     if (effectLoop instanceof EffectLoop(Delay(var timeInMs), Reverb(_
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```

Optimization

```
 1 String apply(Effect effect) {
 2     return switch(effect) {
 3         case Delay(int timeInMs) -> String.format("Delay active of %d
 4         case Reverb(String name, int roomSize) -> String.format("Reve
 5         case Overdrive(int gain) -> String.format("Overdrive active w
 6         case Tremolo(int depth, int rate) -> String.format("Tremolo a
 7         case Tuner(int pitchInHz) -> String.format("Tuner active with
 8         case EffectLoop(Set<Effect> effects) -> effects.stream().map(
 9         default -> String.format("Unknown effect active: %s.", effect
10     };
11 }
```

Optimization

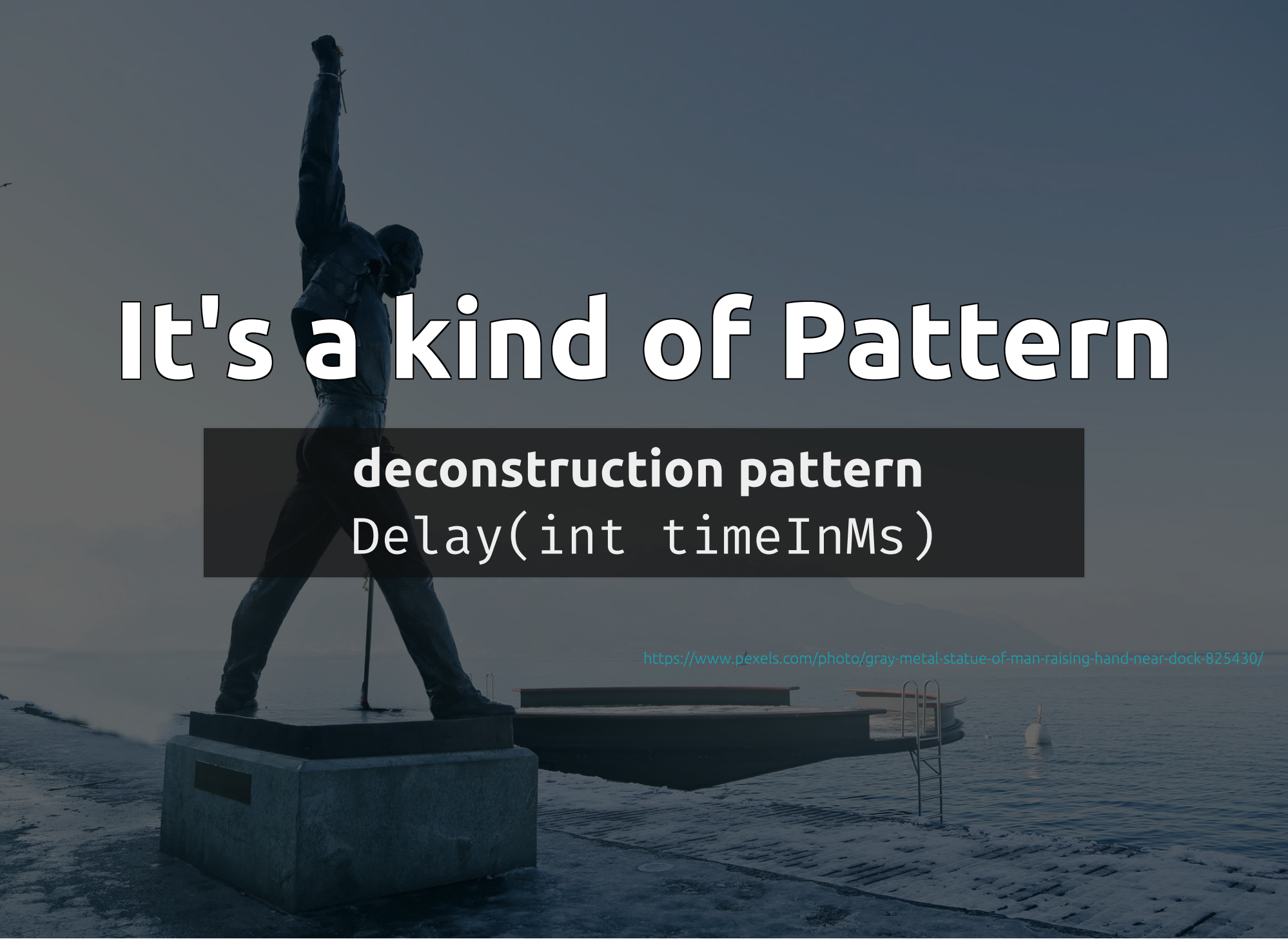
```
1 String apply(Effect effect) {
2     return switch(effect) {
3         // ...
4         case EffectLoop(Set<Effect> effects) -> effects.stream().map(t
5         default -> String.format("Unknown effect active: %s.", effect)
6     };
7 }
```

Optimization

```
1 String apply(Effect effect) {
2     return switch(effect) {
3         // ...
4         case EffectLoop(Tuner(int pitchInHz), _) -> String.format("The
5         case EffectLoop(Set<Effect> effects) -> effects.stream().map(t
6         default -> String.format("Unknown effect active: %s.", effect)
7     };
8 }
```

Benefits

- Better encapsulation
  a case branch only receives data that it actually references.
- More elegant logic
  by using pattern composition
- Optimization
  through the use of any patterns


It's a kind of Pattern
deconstruction pattern
Delay(int timeInMs)
https://www.pexels.com/photo/gray-metal-statue-of-man-raising-hand-near-dock-825430/


It's a kind of Pattern
var pattern
var timeInMs
https://www.pexels.com/photo/gray-metal-statue-of-man-raising-hand-near-dock-825430/

It's a kind of Pattern

any pattern

_

https://www.pexels.com/photo/gray-metal-statue-of-man-raising-hand-near-dock-825430/

Feature Status

Java version	Feature status	JEP
n/a	Exploratory document	Pattern Matching for Java

https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html


Pattern Matching Plays Nice With
Sealed Types and Records
https://pxhere.com/en/photo/752901

Demo

- Convert effect classes to a record
- Acquire constructor, accessor methods etc.

https://pxhere.com/en/photo/1458897

Records

Input:

- Commit to the class being a transparent carrier for its data.

Records

Input:

- Commit to the class being a transparent carrier for its data.

Output:

- constructors
- accessor methods
- `equals()`-implementation
- `hashCode()`-implementation
- `toString()`-implementation
- deconstruction pattern

Demo

- Make `Effect` a sealed type
- Make subclasses `final`, `sealed` or `non-sealed`

https://pxhere.com/en/photo/1458897

Exhaustiveness

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d
4          case Reverb(String name, int roomSize) -> String.format("Reve
5          case Overdrive(int gain) -> String.format("Overdrive active w
6          case Tremolo(int depth, int rate) -> String.format("Tremolo a
7          case Tuner(int pitchInHz) -> String.format("Tuner active with
8          case EffectLoop(Tuner(int pitchInHz), _) -> String.format("Th
9          case EffectLoop(Set<Effect> effects) -> effects.stream().map(
10         default -> String.format("Unknown effect active: %s.", effect
11     };
12 }
```

Exhaustiveness

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d
4          case Reverb(String name, int roomSize) -> String.format("Reve
5          case Overdrive(int gain) -> String.format("Overdrive active w
6          case Tremolo(int depth, int rate) -> String.format("Tremolo a
7          case Tuner(int pitchInHz) -> String.format("Tuner active with
8          case EffectLoop(Tuner(int pitchInHz), _) -> String.format("Th
9          case EffectLoop(Set<Effect> effects) -> effects.stream().map(
10     };
11 }
```

https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html

Feature Status

Records

Java version	Feature status	JEP
14	Preview	JEP 359
15	Second preview	JEP 384
16	Final	JEP 395

Feature Status

Sealed Types

Java version	Feature status	JEP
15	Preview	JEP 360
16	Second preview	JEP 397
17	Final	...


A Better
Serialization
?

Here be dragons!

We can't be sure **at all** that the following features will appear in Java as depicted. They can change a **lot** in the meantime.

https://www.pexels.com/photo/dragon-festival-during-nighttime-6068535/

Opposites

Deconstruction pattern

- transforms an object into a set of typed fields

Constructor

- transforms a set of typed fields into an object

Serialization

- very important feature
- but many people hate its current implementation

Drawbacks

- it undermines the accessibility model
- serialization logic is not 'readable code'
- it bypasses constructors and data validation

Serialization

```
1 public class EffectLoop implements Effect {
2     private String name;
3     private Set<Effect> effects;
4
5     public EffectLoop(String name) {
6         this.name = name;
7         this.effects = new HashSet<>();
8     }
9 }
```

Serialization

```
1  public class EffectLoop implements Effect {
2      private String name;
3      private Set<Effect> effects;
4
5      public EffectLoop(String name) {
6          this.name = name;
7          this.effects = new HashSet<>();
8      }
9
10     public pattern EffectLoop(String name, Effect[] effects) {
11         name = this.name;
12         effects = this.effects.toArray();
13     }
14 }
```

Serialization

```
1  public class EffectLoop implements Effect {
2      private String name;
3      private Set<Effect> effects;
4
5      public EffectLoop(String name) {
6          this.name = name;
7          this.effects = new HashSet<>();
8      }
9
10     public static Effectloop deserialize(String name, Effect[] effect
11         EffectLoop effectLoop = new EffectLoop(name);
12         for (Effect effect : effects) {
13             this.effects.add(effect);
14         }
15         return effectLoop;
16     }
```

Serialization

```
1  public class EffectLoop implements Effect {
2      private String name;
3      private Set<Effect> effects;
4
5      public EffectLoop(String name) {
6          this.name = name;
7          this.effects = new HashSet<>();
8      }
9
10     @Deserializer
11     public static Effectloop deserialize(String name, Effect[] effect
12         EffectLoop effectLoop = new EffectLoop(name);
13         for (Effect effect : effects) {
14             this.effects.add(effect);
15         }
16     }
```

Some challenges remain

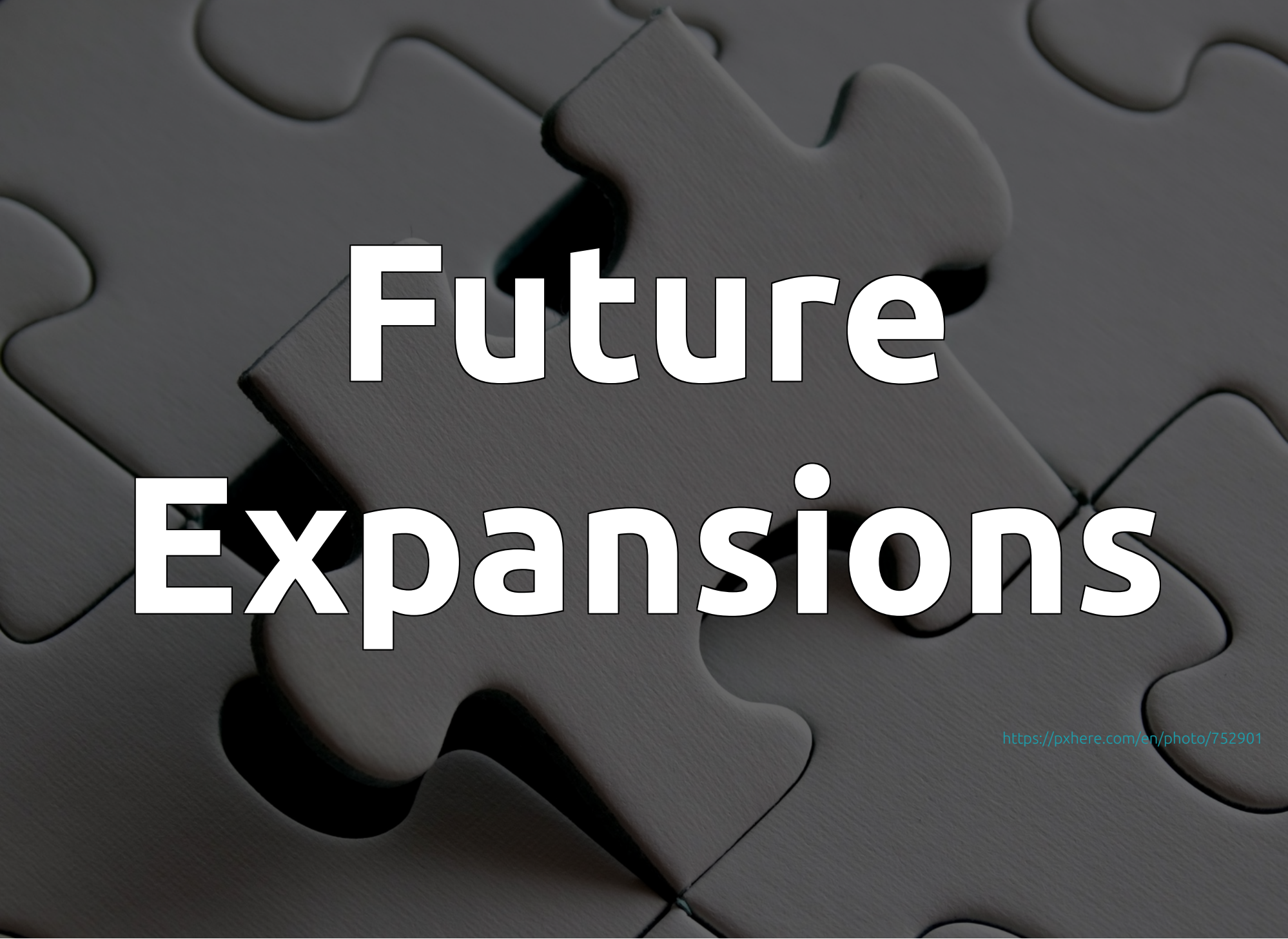
Q: How to support multiple versions of one class?

A: `@Serializer` and `@Deserializer` annotations could get a property `version` in the future.

Feature Status

Java version	Feature status	JEP
n/a	Exploratory document	Towards Better Serialization

https://cr.openjdk.java.net/~briangoetz/amber/serialization.html


Future
Expansions
https://pxhere.com/en/photo/752901

Here be super dragons!

We can't be sure that the following features will appear in Java as depicted, **if at all**.
Proceed with caution!

https://www.pexels.com/photo/dragon-festival-during-nighttime-6068535/

Pattern bind statements

```
1 var reverb = new Reverb("ChamberReverb", 2);
2
3 __let Reverb(String name, int roomSize) = reverb;
4
5 // do something with name & roomSize
```

https://cr.openjdk.java.net/~briangoetz/amber/pattern-match.html

Pattern bind statements

```
1 var reverb = new Reverb("ChamberReverb", 2);
2
3 __let Reverb(String name, int roomSize) = reverb;
4 else throw new IllegalArgumentException("not a Reverb!");
5
6 // do something with name & roomSize
```

Guards

```
1 String apply(Effect effect, Guitar guitar) {
2     return switch(effect) {
3         // (...)
4         case Tremolo tr-> String.format("Tremolo active with depth %d
5         case Tuner tu -> String.format("Tuner active with pitch %d. Mu
6         case EffectLoop el -> el.getEffects().stream().map(this::apply
7         default -> String.format("Unknown effect active: %s.", effect)
8     };
9 }
```

https://openjdk.java.net/jeps/8213076

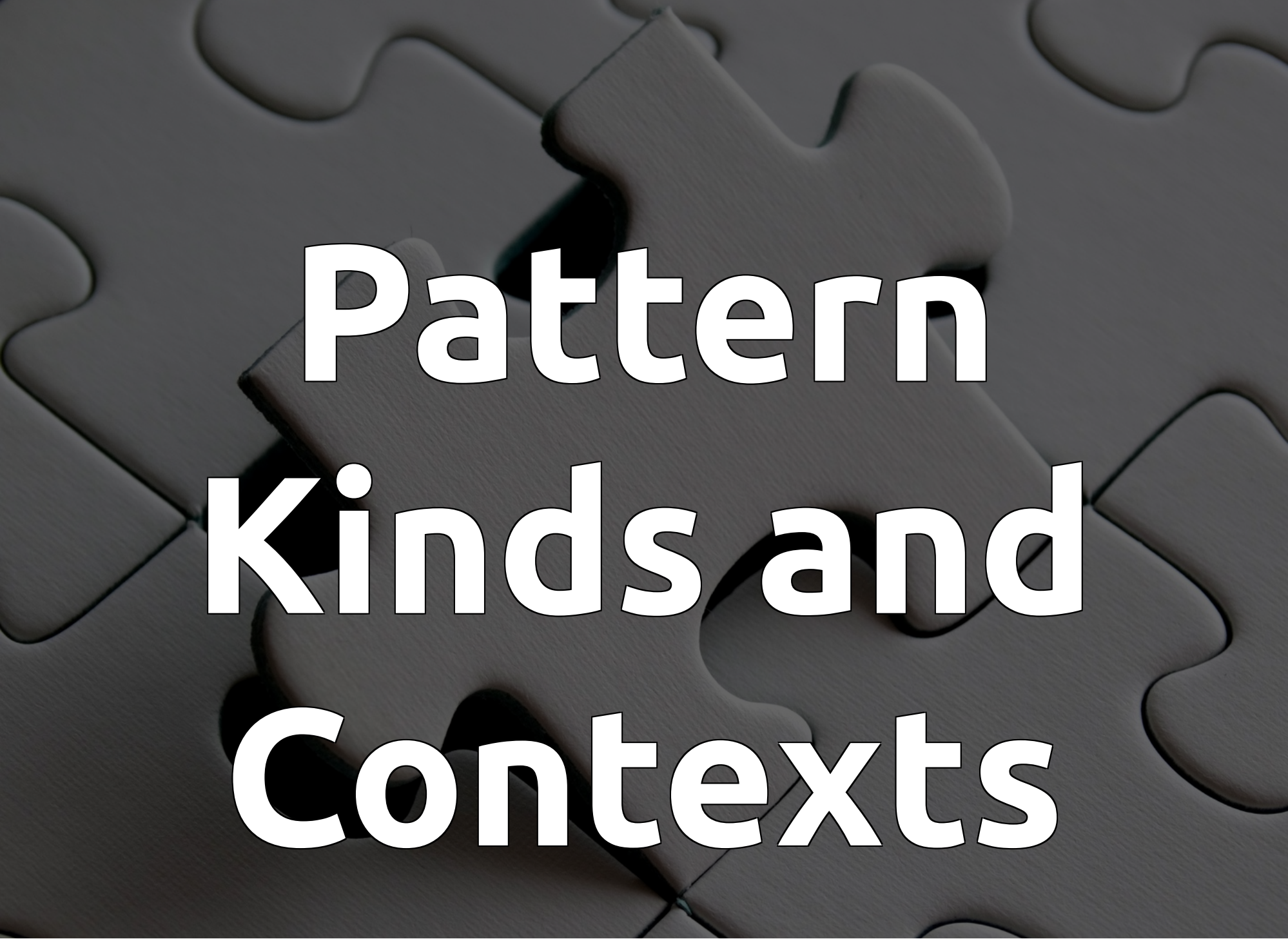
Guards

```
1 String apply(Effect effect, Guitar guitar) {
2     return switch(effect) {
3         // (...)
4         case Tremolo tr-> String.format("Tremolo active with depth %d
5         case Tuner tu && !tu.isInTune(guitar) -> String.format("Guitar
6         case EffectLoop el -> el.getEffects().stream().map(this::apply
7         default -> String.format("Unknown effect active: %s.", effect)
8     };
9 }
```

Other ideas

- Array patterns
- Varargs patterns
- AND patterns
- Patterns in `catch` clauses
- Collection patterns
- Record patterns

https://mail.openjdk.java.net/pipermail/amber-spec-experts/2021-January/002758.html


Pattern
Kinds and
Contexts

Pattern Kinds

Pattern kind	Example	Purpose
type pattern	`Guitar lesPaul`	Perform an instanceof test, cast the target, and bind it to a pattern variable.
constant pattern	`GuitarType.TELECASTER`	Test the target for equality with a constant.
deconstruction pattern	`Delay(int timeInMs)`	Perform an instanceof test, cast the target, destructure the target and recursively match the components to subpatterns.
var pattern	`var timeInMs`	Match anything and bind its target.
any pattern	`_`	Match anything, but bind nothing.

Pattern Contexts

Pattern context	Example	Purpose
instanceof predicate	`product instanceof Guitar guitar`	Test if target matches the indicated pattern.

switch statement or expression	`switch (effect) {`
	`  case Delay d →`
	`}`
		Test if target matches one (or more) of the indicated patterns.

bind statement	`__let Reverb(var name, var roomSize) = reverb;`	Destructure a target using a pattern.

https://pxhere.com/en/photo/752901

Pattern matching...

- is a rich feature arc that will play out over several versions.
- allows us to use type patterns in instanceof.
- improves switch expressions.
- makes destructuring objects as easy (and more similar to) constructing them.
- holds the potential to simplify and streamline much of the code we write today.


Major Feature

Thank you! ☺️

bit.do/javaland-pattern-matching

---

github.com/hannotify/pattern-matching-music-store

---

hannotify.github.io

@hannotify