Pattern Matching

Small Enhancement or Major Feature?

Hanno Embregts @hannotify

Peter Wessels @PeterWessels

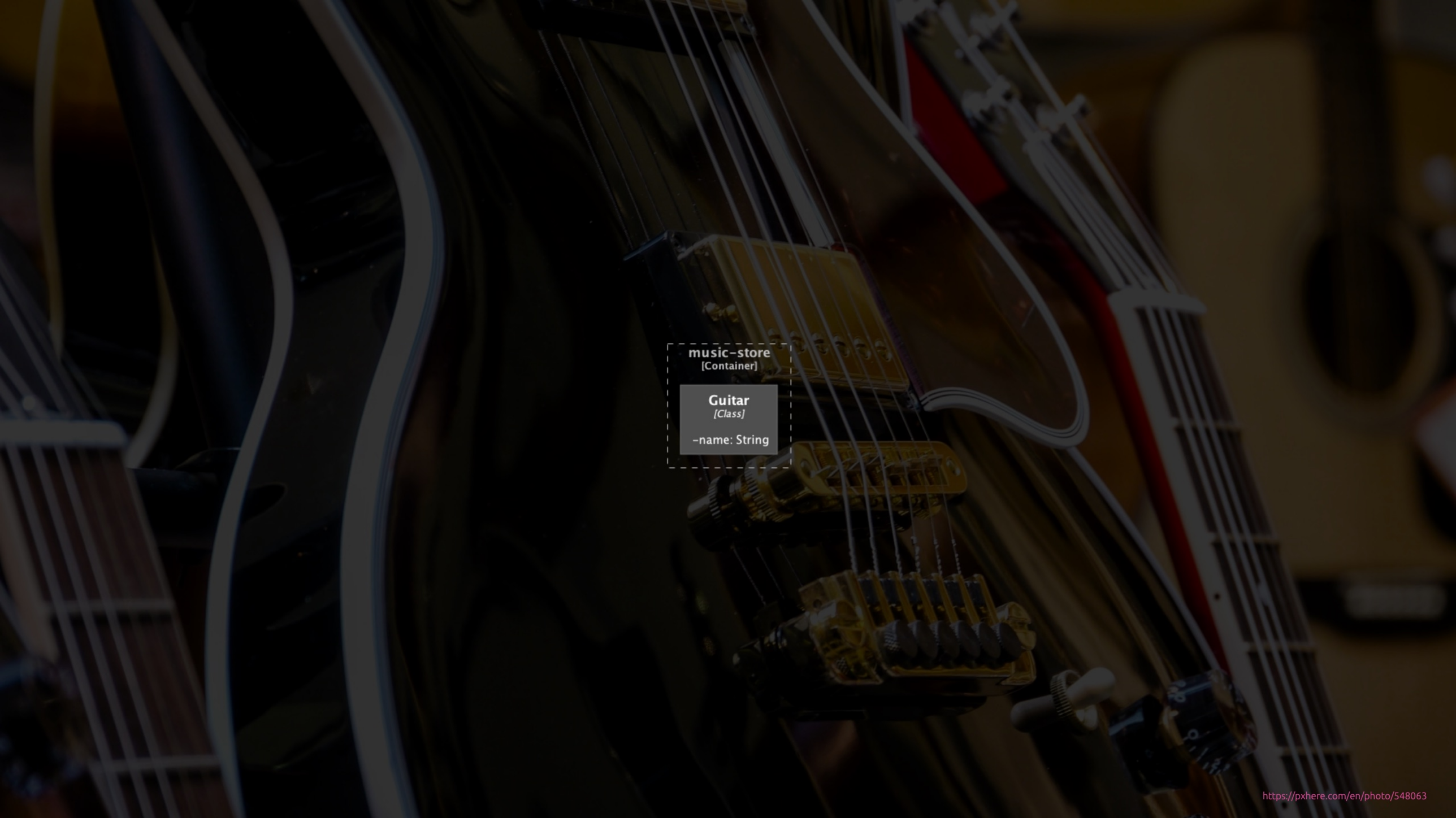
DEVOXX

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

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

Pattern Matching for `instanceof`

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 }
```

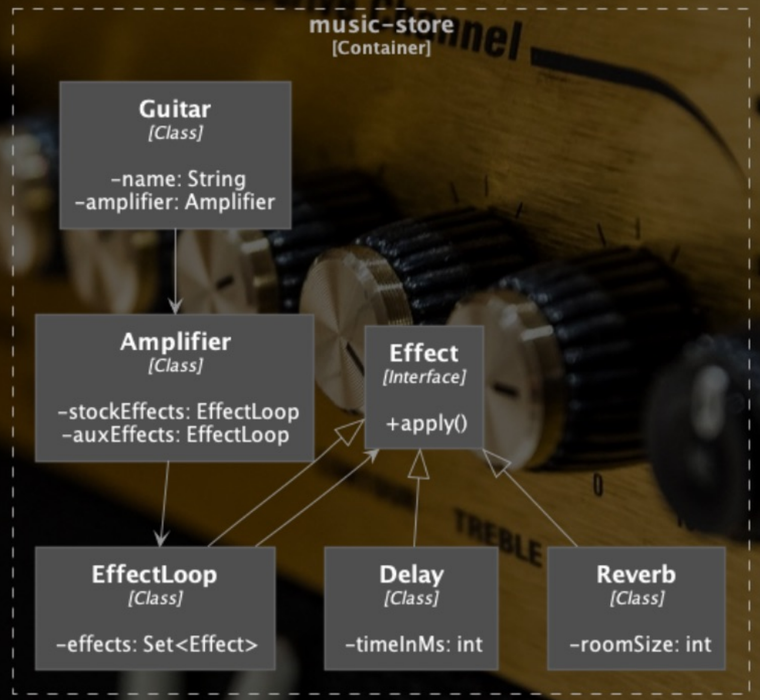
Scoping

Pattern binding variable ('flow scoping')

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

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

https://pxhere.com/en/photo/544037



https://pxhere.com/en/photo/853260

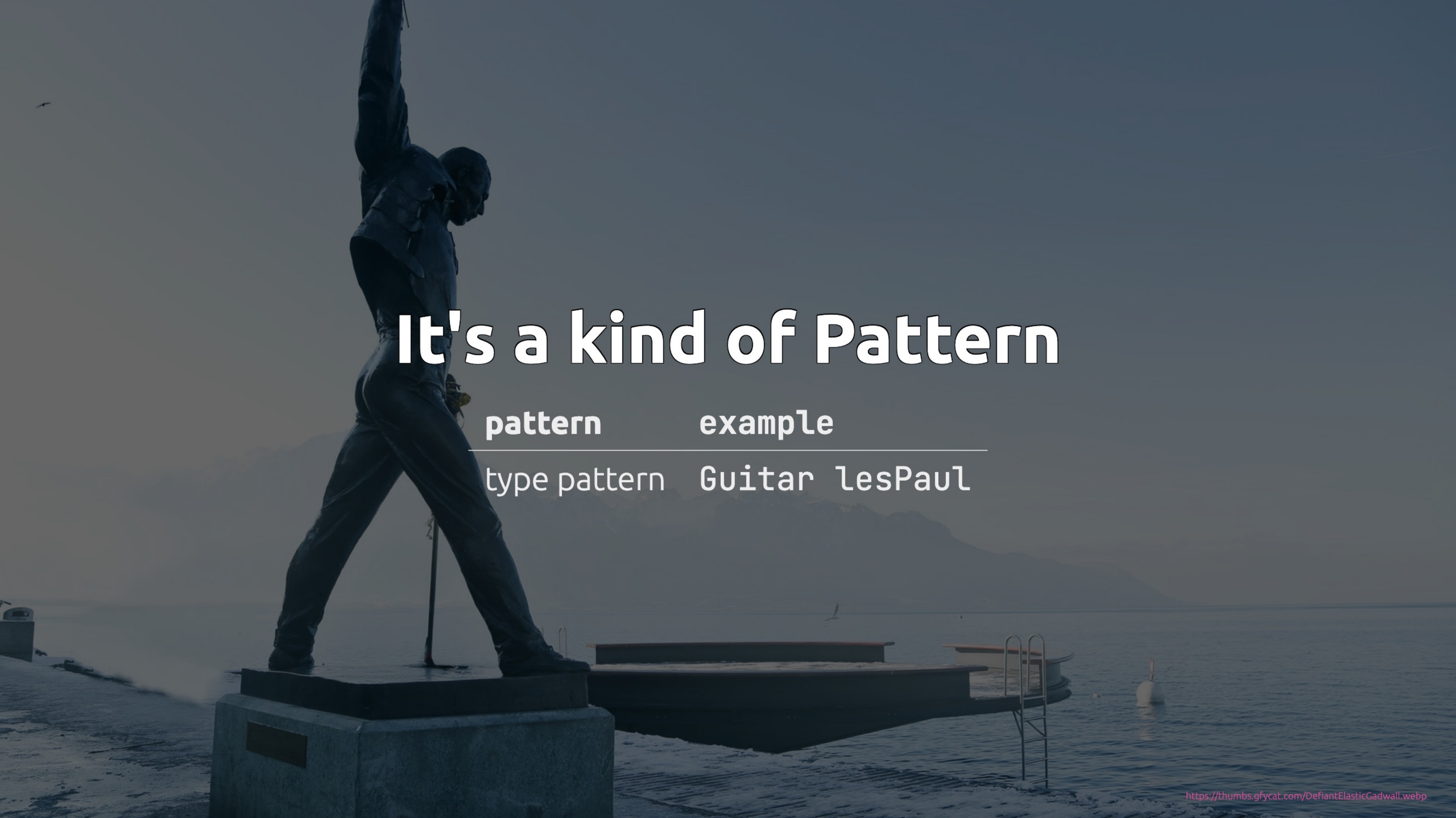
Demo

- Simplify implementation of `equals`

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

Benefits

- Nearly 100% of casts will just disappear!
- More concise



It's a kind of Pattern

pattern	example
type pattern	`Guitar lesPaul`

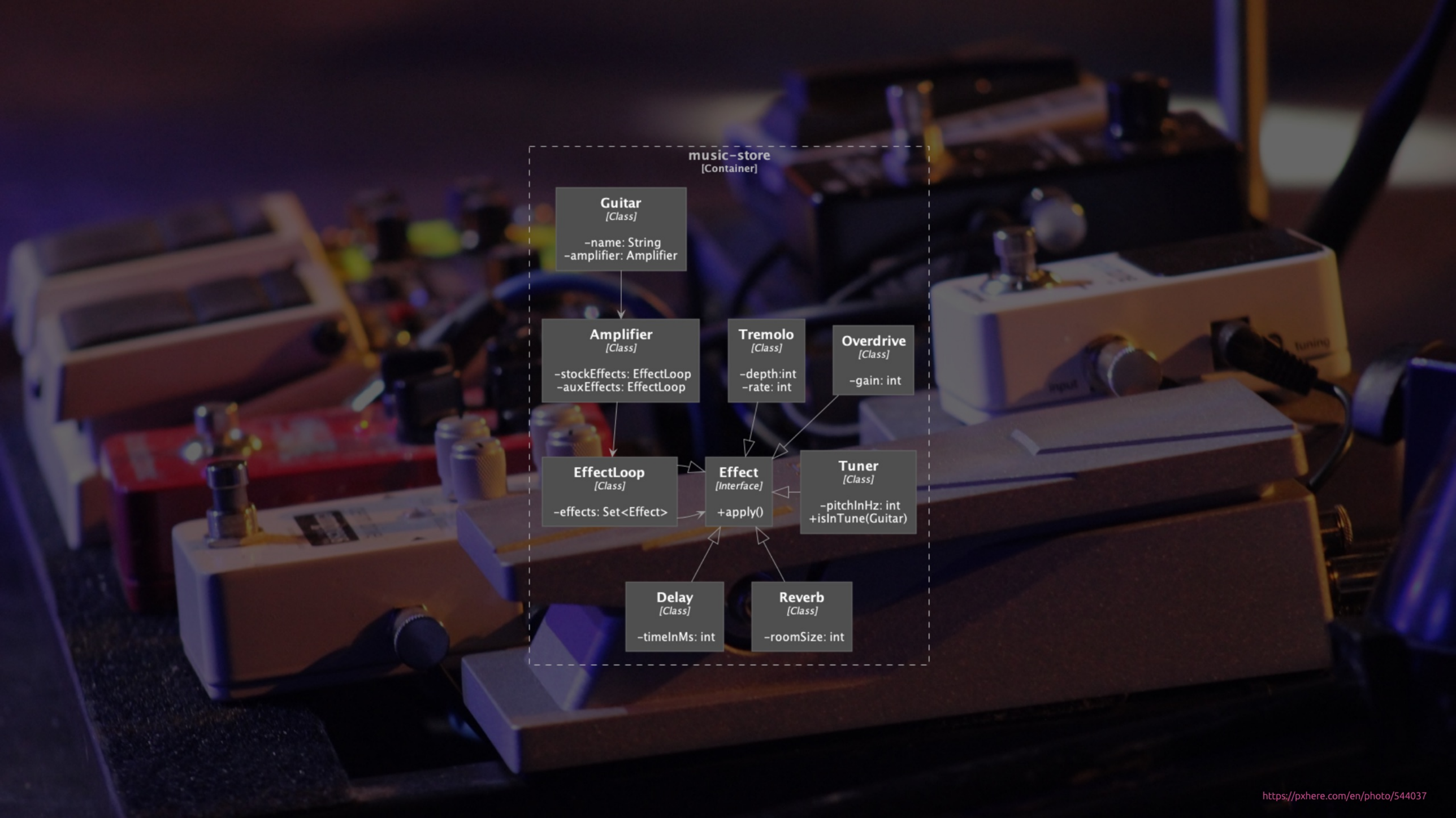
https://thumbs.gfycat.com/DefiantElasticGadwall.webp

Feature Status

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

Pattern Matching for `switch`

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



https://pxhere.com/en/photo/544037

```java
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay) {
        Delay de = (Delay) effect;
        formatted = String.format("Delay active of %d ms.", de.getTimeInMs());
    } else if (effect instanceof Reverb) {
        Reverb re = (Reverb) effect;
        formatted = String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
    } else if (effect instanceof Overdrive) {
        Overdrive ov = (Overdrive) effect;
        formatted = String.format("Overdrive active with gain %d.", ov.getGain());
    } else if (effect instanceof Tremolo) {
        Tremolo tr = (Tremolo) effect;
        formatted = String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
    } else if (effect instanceof Tuner) {
```

```
5         formatted = String.format("Delay active of %d ms.", de.getTimeInMs());
6     } else if (effect instanceof Reverb) {
7         Reverb re = (Reverb) effect;
8         formatted = String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
9     } else if (effect instanceof Overdrive) {
10        Overdrive ov = (Overdrive) effect;
11        formatted = String.format("Overdrive active with gain %d.", ov.getGain());
12    } else if (effect instanceof Tremolo) {
13        Tremolo tr = (Tremolo) effect;
14        formatted = String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
15    } else if (effect instanceof Tuner) {
16        Tuner tu = (Tuner) effect;
17        formatted = String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
18    } else if (effect instanceof EffectLoop) {
19        EffectLoop el = (EffectLoop) effect;
```

```java
String apply(Effect effect) {
    String formatted = "";
    if (effect instanceof Delay de) {
        formatted = String.format("Delay active of %d ms.", de.getTimeInMs());
    } else if (effect instanceof Reverb re) {
        formatted = String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
    } else if (effect instanceof Overdrive ov) {
        formatted = String.format("Overdrive active with gain %d.", ov.getGain());
    } else if (effect instanceof Tremolo tr) {
        formatted = String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
    } else if (effect instanceof Tuner tu) {
        formatted = String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
    } else if (effect instanceof EffectLoop el) {
        formatted = el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
    } else {
        formatted = String.format("Unknown effect active: %s.", effect);
```

```java
    if (effect instanceof Delay de) {
        formatted = String.format("Delay active of %d ms.", de.getTimeInMs());
    } else if (effect instanceof Reverb re) {
        formatted = String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
    } else if (effect instanceof Overdrive ov) {
        formatted = String.format("Overdrive active with gain %d.", ov.getGain());
    } else if (effect instanceof Tremolo tr) {
        formatted = String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
    } else if (effect instanceof Tuner tu) {
        formatted = String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
    } else if (effect instanceof EffectLoop el) {
        formatted = el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
    } else {
        formatted = String.format("Unknown effect active: %s.", effect);
    }
```

Switch expression

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

Switch expression

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

Switch expression

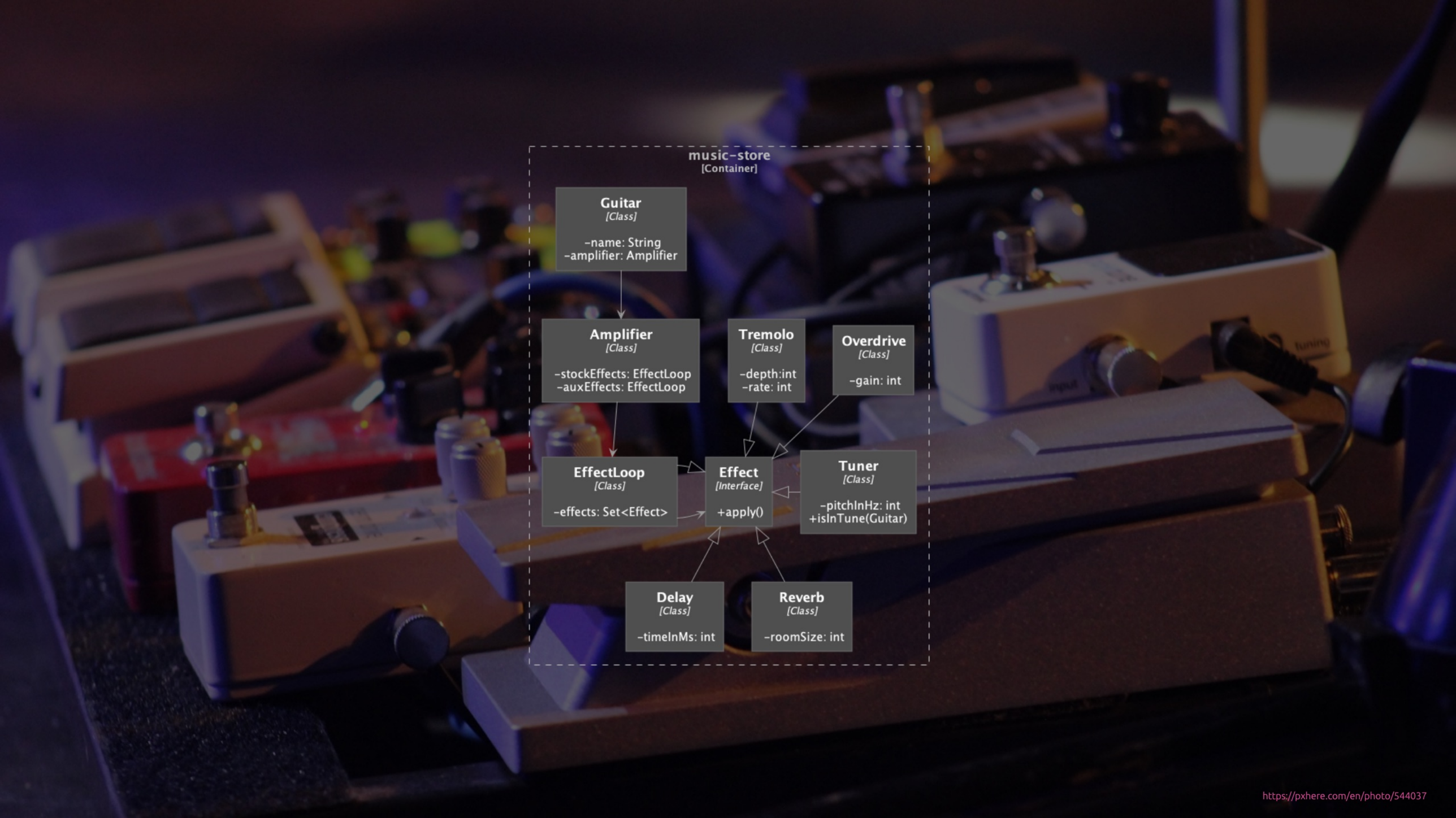
```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          default            → String.format("Unknown effect active: %s.", effect);
9      };
10 }
```

Switch expression

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Switch expression

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```



https://pxhere.com/en/photo/544037

Sensible operations to the effect loop

- `apply()`
- `setVolume(int volume)`
- `contains(Effect... effect)`

Nonsensical operations to the effect loop

- `isTunerActive()`
- `isDelayTimeEqualToReverbRoomSize()`
- `isToneSuitableToPlayPrideInTheNameOfLove()`

Visitor pattern

```
interface EffectVisitor<T> {
    T visit(Tuner effect);
    T visit(Delay effect);
    T visit(Reverb effect);
    // ...
}
class IsTunerActiveVisitor implements EffectVisitor<Boolean> {
    public Boolean visit(Tuner effect) {
        return !effect.isInTune();
    }

    public Boolean visit(Delay effect) { return false; }
    public Boolean visit(Reverb effect) { return false; }
    // ...
}
```

Switch expression

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Switch expression

```
1  static String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Benefits of pattern matching

- No need for the Visitor pattern or a common supertype
- A single expression instead of many assignments
- Less error-prone (in adding cases)
- More concise
- Safer - the compiler can check for missing cases

But what if `effect` is `null`?

```
1  static String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Demo

- What if `effect` is `null`?
  - Solution #1: defensive testing
  - Solution #2: integrate null check in switch
  - Solution #3: combining case labels
- Guarded patterns

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

Guarded patterns

```
1 String apply(Effect effect, Guitar guitar) {
2     return switch(effect) {
3         // (...)
4         case Tremolo tr -> String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
5         case Tuner tu -> String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
6         case EffectLoop el -> el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
7         default -> String.format("Unknown effect active: %s.", effect);
8     };
9 }
```

Guarded patterns

```
1 String apply(Effect effect, Guitar guitar) {
2     return switch(effect) {
3         // (...)
4         case Tremolo tr -> String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
5         case Tuner tu && !tu.isInTune(guitar) -> String.format("Guitar is in need of tuning - tuner active with pitch %d. Muting
6         case EffectLoop el -> el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
7         default -> String.format("Unknown effect active: %s.", effect);
8     };
9 }
```

Guarded patterns

```
1 switch(effect) {
2     // ...
3     case Tuner tu:
4         if (!tu.isInTune(guitar)) { // tuning is needed }
5         else { // no tuning is needed }
6         break;
7     // ...
8 }
```

It's a kind of Pattern

pattern	example
type pattern	`Guitar lesPaul`
guarded pattern	`Tuner tu && !tu.isInTune(guitar)`

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

Feature Status

Java version	Feature status	JEP
17	Preview	JEP 406
18	Second preview (proposed to target)	JEP 420

Deconstruction Patterns

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

Disclaimer

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

https://pxhere.com/en/photo/1359311

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive ov  → String.format("Overdrive active with gain %d.", ov.getGain());
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay de      → String.format("Delay active of %d ms.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive(int gain) → String.format("Overdrive active with gain %d.", gain);
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
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.", de.getTimeInMs());
4          case Reverb re     → String.format("Reverb active of type %s and roomSize %d.", re.getName(), re.getRoomSize());
5          case Overdrive(int gain) → String.format("Overdrive active with gain %d.", gain);
6          case Tremolo tr    → String.format("Tremolo active with depth %d and rate %d.", tr.getDepth(), tr.getRate());
7          case Tuner tu      → String.format("Tuner active with pitch %d. Muting all signal!", tu.getPitchInHz());
8          case EffectLoop el → el.getEffects().stream().map(this::apply).collect(Collectors.joining(System.lineSeparator()));
9          default            → String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Deconstruction patterns

```
1  String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d ms.", timeInMs);
4          case Reverb(String name, int roomSize) -> String.format("Reverb active of type %s and roomSize %d.", name, roomSize);
5          case Overdrive(int gain) -> String.format("Overdrive active with gain %d.", gain);
6          case Tremolo(int depth, int rate) -> String.format("Tremolo active with depth %d and rate %d.", depth, rate);
7          case Tuner(int pitchInHz) -> String.format("Tuner active with pitch %d. Muting all signal!", pitchInHz);
8          case EffectLoop(Set<Effect> effects) -> effects.stream().map(this::apply).collect(Collectors.joining(System.lineSeparat
9          default -> String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Pattern composition

```
1 static boolean isDelayTimeEqualToReverbRoomSize(EffectLoop effectLoop) {
2
3 }
```

Pattern composition

```
1 static boolean isDelayTimeEqualToReverbRoomSize(EffectLoop effectLoop) {
2     if (effectLoop instanceof EffectLoop(Delay(int timeInMs), Reverb(String name, int roomSize))) {
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```

Pattern composition

```
1  static boolean isDelayTimeEqualToReverbRoomSize(EffectLoop effectLoop) {
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 }
```

It's a kind of Pattern

pattern	example
type pattern	`Guitar lesPaul`
guarded pattern	`Tuner tu && !tu.isInTune(guitar)`
deconstruction pattern	`Delay(int timeInMs)`

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

Var and any patterns

```
1 // Pre-Java 10
2 Guitar telecaster = new Guitar("Fender Telecaster Baritone Blacktop", GuitarType.TELECASTER);
3
4 // Java 10
5 var telecaster = new Guitar("Fender Telecaster Baritone Blacktop", GuitarType.TELECASTER);
```

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

Var and any patterns

```
1 static boolean isDelayTimeEqualToReverbRoomSize(EffectLoop effectLoop) {
2     if (effectLoop instanceof EffectLoop(Delay(int timeInMs), Reverb(String name, int roomSize))) {
3         return timeInMs == roomSize;
4     }
5     return false;
6 }
```

Var and any patterns

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

http://gph.is/2lFlHIK

Var and any patterns

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

Optimization

```
1  static String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d ms.", timeInMs);
4          case Reverb(String name, int roomSize) -> String.format("Reverb active of type %s and roomSize %d.", name, roomSize);
5          case Overdrive(int gain) -> String.format("Overdrive active with gain %d.", gain);
6          case Tremolo(int depth, int rate) -> String.format("Tremolo active with depth %d and rate %d.", depth, rate);
7          case Tuner(int pitchInHz) -> String.format("Tuner active with pitch %d. Muting all signal!", pitchInHz);
8          case EffectLoop(var effects) -> effects.stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()))
9          default -> String.format("Unknown effect active: %s.", effect);
10     };
11 }
```

Optimization

```
1 static String apply(Effect effect) {
2     return switch(effect) {
3         // ...
4         case EffectLoop(var effects) -> effects.stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()));
5         default -> String.format("Unknown effect active: %s.", effect);
6     };
7 }
```

Optimization

```
1 static String apply(Effect effect) {
2     return switch(effect) {
3         // ...
4         case EffectLoop(Tuner(int pitchInHz), _) -> String.format("The EffectLoop contains a tuner with pitch %d. Muting all sig
5         case EffectLoop(var effects) -> effects.stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()));
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

pattern	example
type pattern	`Guitar lesPaul`
guarded pattern	`Tuner tu && !tu.isInTune(guitar)`
deconstruction pattern	`Delay(int timeInMs)`
var pattern	`var timeInMs`

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

It's a kind of Pattern

pattern	example
type pattern	`Guitar lesPaul`
guarded pattern	`Tuner tu && !tu.isInTune(guitar)`
deconstruction pattern	`Delay(int timeInMs)`
var pattern	`var timeInMs`
any pattern	`_`

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

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

Completeness on pattern switch expressions

```
1 static String apply(Object object) {
2     return switch(object) {
3         case Effect e -> e.apply();
4         case String s -> String.format("Tried to apply a String: %s", s);
5         case Number n -> String.format("Tried to apply a Number: %d", n);
6     };
7 }
```

Completeness on pattern switch expressions

```
1 static String apply(Object object) {
2     return switch(object) { // Won't compile - incomplete!
3         case Effect e -> e.apply();
4         case String s -> String.format("Tried to apply a String: %s", s);
5         case Number n -> String.format("Tried to apply a Number: %d", n);
6     };
7 }
```

Completeness on pattern switch expressions

```
1 static String apply(Object object) {
2     return switch(object) {
3         case Effect e -> e.apply();
4         case String s -> String.format("Tried to apply a String: %s", s);
5         case Number n -> String.format("Tried to apply a Number: %d", n);
6         default       -> String.format("Tried to apply an Object: %s", object);
7     };
8 }
```

Demo

- Make `Effect` a sealed type

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

Sealed types yield completeness

```
1  static String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d ms.", timeInMs);
4          case Reverb(String name, int roomSize) -> String.format("Reverb active of type %s and roomSize %d.", name, roomSize);
5          case Overdrive(int gain) -> String.format("Overdrive active with gain %d.", gain);
6          case Tremolo(int depth, int rate) -> String.format("Tremolo active with depth %d and rate %d.", depth, rate);
7          case Tuner(int pitchInHz) -> String.format("Tuner active with pitch %d. Muting all signal!", pitchInHz);
8          case EffectLoop(Tuner(int pitchInHz), _) -> String.format("The EffectLoop contains a tuner with pitch %d. Muting all si
9          case EffectLoop(var effects) -> effects.stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()))
10         default -> String.format("Unknown effect active: %s.", effect);
11     };
12 }
```

Sealed types yield completeness

```
1  static String apply(Effect effect) {
2      return switch(effect) {
3          case Delay(int timeInMs) -> String.format("Delay active of %d ms.", timeInMs);
4          case Reverb(String name, int roomSize) -> String.format("Reverb active of type %s and roomSize %d.", name, roomSize);
5          case Overdrive(int gain) -> String.format("Overdrive active with gain %d.", gain);
6          case Tremolo(int depth, int rate) -> String.format("Tremolo active with depth %d and rate %d.", depth, rate);
7          case Tuner(int pitchInHz) -> String.format("Tuner active with pitch %d. Muting all signal!", pitchInHz);
8          case EffectLoop(Tuner(int pitchInHz), _) -> String.format("The EffectLoop contains a tuner with pitch %d. Muting all si
9          case EffectLoop(var effects) -> effects.stream().map(Effect::apply).collect(Collectors.joining(System.lineSeparator()))
10     };
11 }
```

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

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

Record patterns

```
1 record Amplifier(String name, EffectLoop stockEffects, EffectLoop auxEffects) { }
2 record EffectLoop(Delay delay, Reverb reverb) { }
```

```
1 static String switchOn(Amplifier amplifier) {}
2     return switch(effectLoop) {
3         // ...
4         case Amplifier(var name, EffectLoop(Delay(int timeInMs), Reverb r), var auxEffects) -> String.format("Stock delay active
5         // ...
6     }
7 }
```

Array patterns

```
1 record EffectLoop(String name, int volume, Effect... effects) { }
```

```
1 static String apply(EffectLoop effectLoop) {}
2     return switch(effectLoop) {
3         case EffectLoop(var name, var volume) → "Effect loop contains no effects.";
4         case EffectLoop(var name, var volume, var effect) → "Effect loop contains exactly one effect.";
5         case EffectLoop(var name, var volume, var effect, ...) → "Effect loop contains more than one effect.";
6         case EffectLoop(var name, var volume, var effect1, var effect2) → "Effect loop contains exactly two effects.";
7         case EffectLoop(var name, var volume, var effect1, var effect2, ...) → "Effect loop contains more than two effects.";
8     }
9 }
```

It's a kind of Pattern

pattern	example
..	..
var pattern	`var timeInMs`
any pattern	`_`
array pattern	`EffectLoop(var name, var effect, ...)`

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

Feature Status

Sealed Types

Java version	Feature status	JEP
15	Preview	JEP 360
16	Second preview	JEP 397
17	Final	JEP 409

Feature Status

Completeness

Java version	Feature status	JEP
17	Preview	JEP 406
18	Second preview (proposed to target)	JEP 420

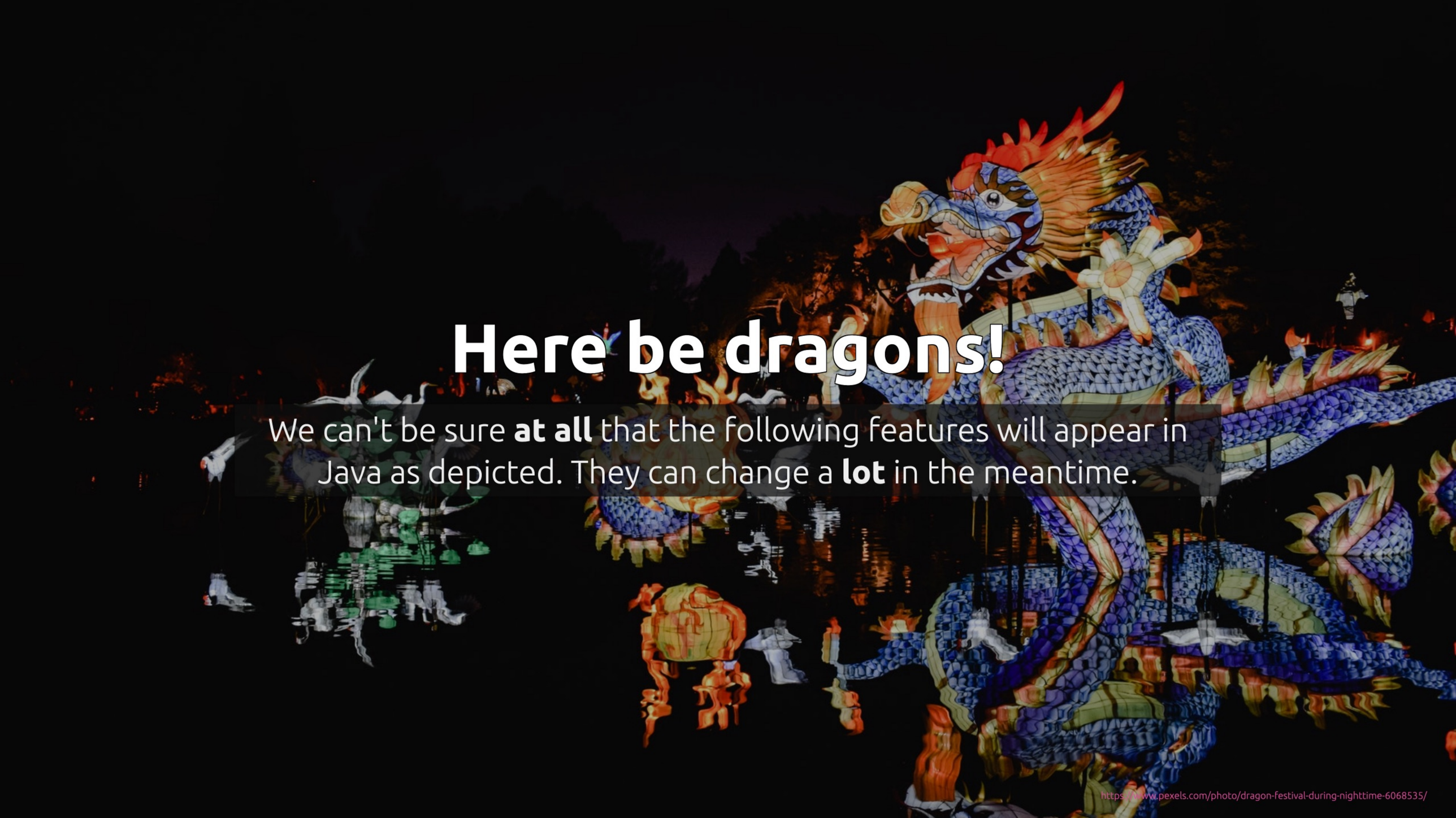
Feature Status

Record & Array Patterns

Java version	Feature status	JEP
n/a	Preview	JEP 405

A Better Serialization?

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/

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

```
5     public EffectLoop(String name) {
6         this.name = name;
7         this.effects = new HashSet<>();
8     }
9
10    public static Effectloop deserialize(String name, Effect[] effects) {
11        EffectLoop effectLoop = new EffectLoop(name);
12        for (Effect effect : effects) {
13            this.effects.add(effect);
14        }
15        return effectLoop;
16    }
17
18    public pattern EffectLoop(String name, Effect[] effects) {
19        name = this.name;
20        effects = this.effects.toArray();
```

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[] effects) {
12         EffectLoop effectLoop = new EffectLoop(name);
13         for (Effect effect : effects) {
14             this.effects.add(effect);
15         }
```

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
```

Other ideas

- Deconstruction patterns for all classes
- Enhanced array patterns
  `String[] { [8] → var eighthElement, [9] → var ninthElement}`
- AND patterns
- Patterns in `catch` clauses
- Collection patterns

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

Pattern Contexts

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

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.

Wrap-up

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 as (and more similar to) constructing them.
- holds the potential to simplify and streamline much of the code we write today.

Major Feature

Thank you! 😀

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

hanno.codes

peterwessels.nl

@hannotify

@PeterWessels