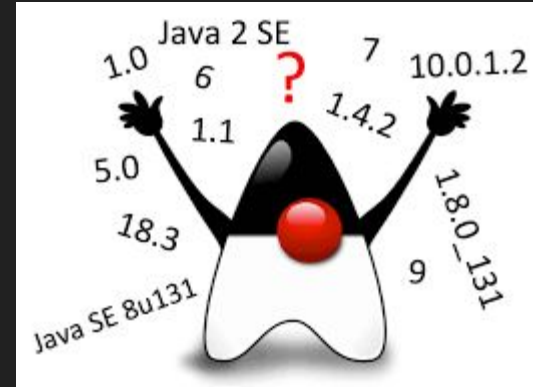




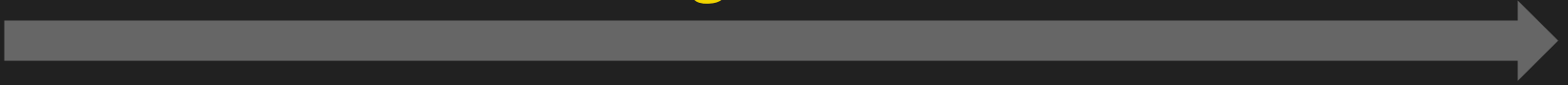
*Ron Veen*



*What happened since Java 11*



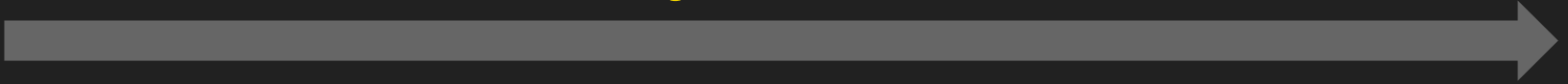
# Agenda



- Java 8 to Java 11
- Java 12 to Java 17



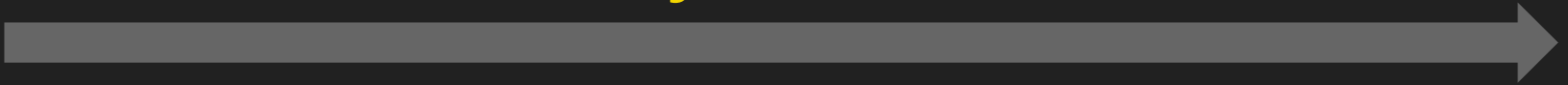
# Java 8



- Lambdas
- Streams



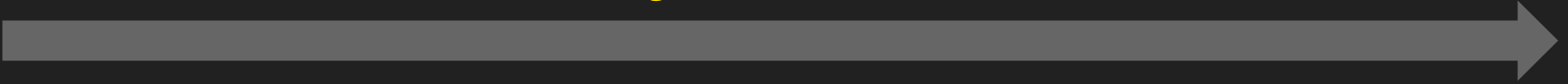
# Java 9



- JVM Modularization
- Http client
- Process API
- Private methods in interfaces



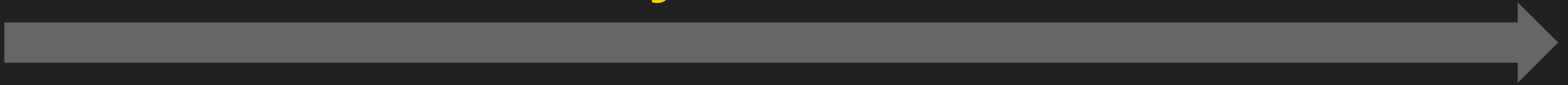
# Java 10



- Local variable type inference



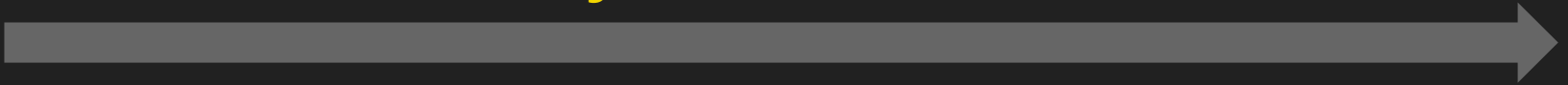
# Java 11



- Type inference for lambda variables
- Flight recorder
- ZGC Garbage Collector



# Java 12 - 17



- Switch expression
- Text blocks
- Records
- Instanceof
- Sealed classes
- Pattern matching for switch



# Java 12 - 17 matrix

	Switch expression
Java 12	Preview
Java 13	2nd Preview
Java 14	Final
Java 15	
Java 16	
Java 17	





# Switch expression



- Label -> syntax
- No fall-through, eliminates the need for a **break**
- The value after the -> is value of the expression
- You can have multiple labels left of the ->
- Must be exhaustive
- Switch Labelled Rule



# Switch expression

```
private void run(int value) {  
    String answer = switch (abs(value)) {  
        case 0 -> "Zero";  
        case 1,2,3,4,5 -> "Five or less";  
        case 42 -> "The answer to life, the universe and everything";  
        default -> "Not sure, but mor than six";  
    };  
    System.out.println(answer);  
}
```



# Switch expression



- Label : syntax
- Eliminates the need for a **break**
- Values are returned via **yield**
- You can have multiple labels on the left of the colon (:)
- Must be exhaustive
- **break** and **continue** are not allowed
- Switch Labelled Statement group



# Switch expression

```
private void run(int value) {  
    String answer = switch (abs(value)) {  
        case 0: yield "Zero";  
        case 1,2,3,4,5: yield "Five or less";  
        case 42: yield "The answer to life, the universe and everything";  
        default: yield "Not sure, but mor than six";  
    };  
    System.out.println(answer);  
}
```



# Switch expression

```
private void runAnother(int value) {  
    String answer = switch (abs(value)) {  
        case 0:  
            System.out.println("Value is zero");  
            yield "Zero";  
        case 1,2,3,4,5:  
            System.out.println("Value is between 1 and 5");  
            yield "Five or less";  
        case 42:  
            System.out.println("42!!!");  
            yield "The answer to life, the universe and everything";  
        default:  
            System.out.println("Another value");  
            yield "Not sure, but mor than six";  
    };  
    System.out.println(answer);  
}
```



# Java 12 - 17 matrix

	Switch expression	Text blocks
Java 12	Preview	
Java 13	2nd Preview	Preview
Java 14	Final	2nd Preview
Java 15		Final
Java 16		
Java 17		



# Text blocks

- Java String spanning multiple lines
- ```
"Hello Java 17".equals("""  
    Hello Java 17""");
```
- Text block begins with `"""` + line terminator
- You can have `"` within a text block. No more escaping `\`
- Placement of final `"""` is import
  - On the same line: no line terminator
  - On its own line: line terminator (`\n`) gets added



# Text blocks

```
private void run() {  
    System.out.println("""  
        <html>  
            <body>  
                <p>  
                    <div>Finally some text with "</div>  
                """);  
}
```

```
<html>  
    <body>  
        <p>  
            <div>Finally some text with "</div>
```





# Text blocks

```
private void run() {  
    System.out.println("""  
        private void run() {  
            System.out.println("""  
        <html>  
        <body>  
            <p>  
                <div>Finally some text with "</div>  
        """);  
    }
```



# Text blocks

```
private void run() {  
    System.out.println("""  
        <html>  
        <body>  
        <p>  
            <div>Finally some text with "</div>  
        """);  
}
```

```
<html>  
    <body>  
        <p>  
            <div>Finally some text with "</div>
```



# Text blocks

```
private void run() {  
    System.out.println("""  
        line 1  
        line 2          \s"" + "*");  
}
```

```
line 1  
line 2*
```

```
line 1  
line 2      *
```



# Java 12 - 17 matrix

	Switch expression	Text blocks	Records
Java 12	Preview		
Java 13	2nd Preview	Preview	
Java 14	Final	2nd Preview	Preview
Java 15		Final	2nd Preview
Java 16			Final
Java 17			



# Records

- Special kind of a class
- Specified in the header its content
- Generates its constructor, accessors, equals, hashCode and toString methods
- Records are final

- ```
public record SimpleRecord(String name, boolean isStriped) { }
```



# Records

- Compact constructor

```
public record CompactConstructorRecord(String name, String color, int age) {  
  
    public CompactConstructorRecord {  
        requireNonNull(name, message: "Name cannot be null");  
        if (age < 0) {  
            throw new IllegalArgumentException("Age should be 0 or larger");  
        }  
    }  
}
```



# Records

```
public record CompactConstructorRecord(String name, String color, int age) {  
  
    public CompactConstructorRecord {  
        requireNonNull(name, message: "Name cannot be null");  
        if (age < 0) {  
            throw new IllegalArgumentException("Age should be 0 or larger");  
        }  
    }  
  
    public static void main(String... args) {  
        CompactConstructorRecord ccr = new CompactConstructorRecord( name: "Mittens", color: "gray", age: 3);  
        System.out.println(ccr.toString());  
  
        // Next line will throw an exception  
        ccr = new CompactConstructorRecord( name: "Garfield", color: "red", age: -1);  
    }  
}
```

CompactConstructorRecord[name=Mittens, color=gray, age=3]

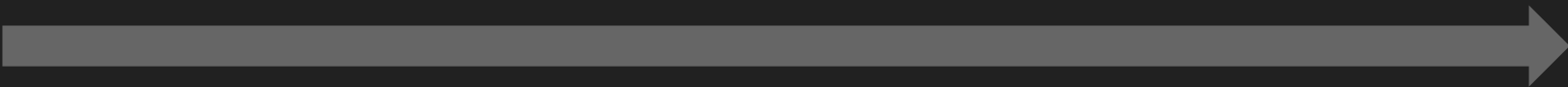
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : Age should be 0 or larger

at CompactConstructorRecord.<init>(CompactConstructorRecord.java:8)

at CompactConstructorRecord.main(CompactConstructorRecord.java:17)



# Records



- Cannot declare additional instance variables
- Cannot extend other classes or records
- Accessors not prefix with name => name() instead of getName()
- Extend interfaces
- Static fields, initializers and methods are allowed
- Add your own instance methods
- Nested classes and nested records are allowed





# Records

```
public record CompactConstructorRecord(String name, String color, int age) implements ActionListener {  
  
    // Static variables are allowed  
    public static int ageSum = 0;  
  
    public CompactConstructorRecord {  
        requireNonNull(name, message: "Name cannot be null");  
        if (age < 0) {  
            throw new IllegalArgumentException("Age should be 0 or larger");  
        }  
        ageSum += age(); // update static var  
    }  
  
    public static void main(String... args) {  
        CompactConstructorRecord ccr = new CompactConstructorRecord( name: "Mittens", color: "gray", age: 3);  
        // Invoke the generated toString()  
        System.out.println(ccr);  
        // Access instance variables  
        System.out.println("name is " + ccr.name());  
        // Access static variables  
        System.out.println("Collected age is " + CompactConstructorRecord.ageSum);  
        // Access instance methods  
        System.out.println("Reversed name is " + ccr.reverseName());  
  
        // Next line will throw an exception  
        ccr = new CompactConstructorRecord( name: "Garfield", color: "red", age: -1);  
    }  
  
    // Instance method  
    public String reverseName() {  
        return new StringBuilder(name()).reverse().toString().toLowerCase();  
    }  
  
    // Implements interface  
    @Override  
    public void actionPerformed(ActionEvent e) {  
    }  
}
```



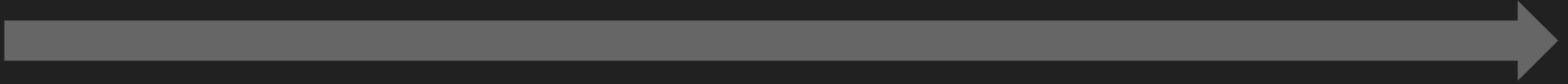
# Records

```
private List<CompactConstructorRecord> getData() {  
    var data = new ArrayList<CompactConstructorRecord>();  
    data.add(new CompactConstructorRecord( name: "Name 1", color: "Red", age: 1));  
    data.add(new CompactConstructorRecord( name: "Name 2", color: "Grey", age: 6));  
    data.add(new CompactConstructorRecord( name: "Name 3", color: "White", age: 2));  
    data.add(new CompactConstructorRecord( name: "Name 4", color: "Green", age: 5));  
    data.add(new CompactConstructorRecord( name: "Name 5", color: "Brown", age: 3));  
    return data;  
}
```

```
private void sortData(List<CompactConstructorRecord> data) {  
  
    record Person(CompactConstructorRecord info, int age) {}  
  
    var result : List<CompactConstructorRecord> = data.stream() Stream<CompactConstructorRecord>  
        .map(d -> new Person(d, d.age())) Stream<Person>  
        .sorted((p1, p2) -> Integer.compare(p2.age(), p1.age()))  
        .map(Person::info) Stream<CompactConstructorRecord>  
        .toList();  
    result.forEach(System.out::println);  
}
```



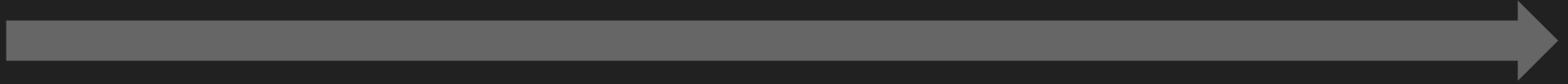
# Records



- Data transfer objects
- Safer serialisation



# Records



- `boolean Class.isRecord ()`
- `RecordComponent[] Class.getRecordComponents()`



# Java 12 - 17 matrix

|         | Switch expression | Text blocks | Records     | instanceof  |
|---------|-------------------|-------------|-------------|-------------|
| Java 12 | Preview           |             |             |             |
| Java 13 | 2nd Preview       | Preview     |             |             |
| Java 14 | Final             | 2nd Preview | Preview     | Preview     |
| Java 15 |                   | Final       | 2nd Preview | 2nd Preview |
| Java 16 |                   |             | Final       | Final       |
| Java 17 |                   |             |             |             |



# Instance of pattern matching

- Pattern matching is a two stage process
- 1 testing (the object structure)
- 2 extracting data from the object
- Pattern
  - Predicate (test)
  - Set of local variables (pattern variables)
- Variables are only extracted if the test is successful



# Instance of pattern matching

```
public interface Animal {  
    void speak();  
}  
  
public class Cat implements Animal {  
  
    @Override  
    public void speak() { System.out.println("Meow"); }  
  
    public void jump() { System.out.println("I'm jumping!"); }  
}  
  
public class Bird implements Animal {  
  
    @Override  
    public void speak() { System.out.println("Twitter"); }  
  
    public void fly() { System.out.println("I'm flying!"); }  
}
```



# InstanceOf pattern matching

```
private void doClassic(Animal animal) {  
    if (animal instanceof Cat) {  
        Cat c = (Cat)animal;  
        c.speak();  
        c.jump();  
    }  
  
    if (animal instanceof Bird) {  
        Bird b = (Bird)animal;  
        b.speak();  
        b.fly();  
    }  
}
```






# Instanceof pattern matching

```
private void doNew(Animal animal) {  
    if (animal instanceof Cat c) {  
        c.speak();  
        c.jump();  
    }  
  
    if (animal instanceof Bird b) {  
        b.speak();  
        b.fly();  
    }  
}
```



# InstanceOf pattern matching

```
private void doNew(Animal animal) {  
    if (animal instanceof Cat c) {  
        c.speak();  
        c.jump();  
    }  
  
    if (animal instanceof Bird b) {  
        b.speak();  
        b.fly();  
    }  
}
```

Two red arrows are present. One arrow points from the right towards the ' instanceof ' text in the first if-statement. The other arrow points from the right towards the variable 'c' in the same if-statement.

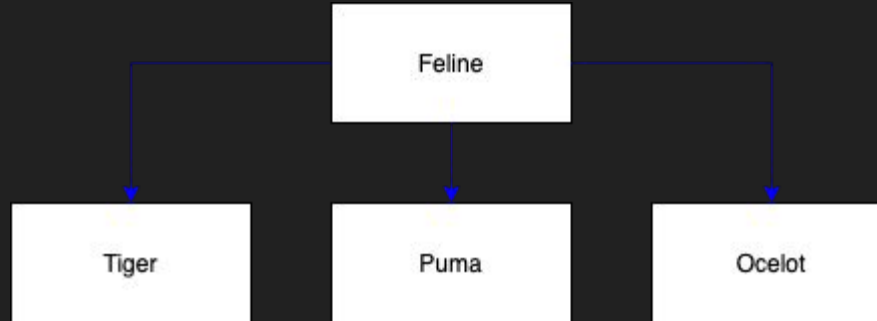


# Java 12 - 17 matrix

|         | Switch expression | Text blocks | Records     | instanceof  | Sealed classes |
|---------|-------------------|-------------|-------------|-------------|----------------|
| Java 12 | Preview           |             |             |             |                |
| Java 13 | 2nd Preview       | Preview     |             |             |                |
| Java 14 | Final             | 2nd Preview | Preview     | Preview     |                |
| Java 15 |                   | Final       | 2nd Preview | 2nd Preview | Preview        |
| Java 16 |                   |             | Final       | Final       | 2nd Preview    |
| Java 17 |                   |             |             |             | Final          |

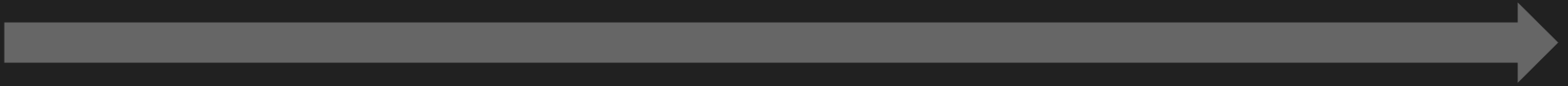


# Sealed classes





# Sealed classes



```
public sealed interface Feline {  
    public sealed interface Feline permits Tiger, Puma, Ocelot {  
    }  
}
```



# Sealed classes

- Subclasses need to be accessible to the sealed class during compile time
- Subclasses need to directly extend the sealed class
- Subclasses must have one modifier of
  - final
  - sealed
  - non-sealed
- Subclasses need to be in the same module (or package) as the sealed class
- Records are allowed in the permits clause



# Sealed classes

- Sealed classes can be abstract
- `boolean Class.isSealed( )`
- `Class<?> Class.permittedSubclasses ( )`
- Link with pattern matching for exhaustive analysis of patterns



# Java 12 - 17 matrix

|         | Switch expression | Text blocks | Records     | instanceof  | Sealed classes | Pattern matching |
|---------|-------------------|-------------|-------------|-------------|----------------|------------------|
| Java 12 | Preview           |             |             |             |                |                  |
| Java 13 | 2nd Preview       | Preview     |             |             |                |                  |
| Java 14 | Final             | 2nd Preview | Preview     | Preview     |                |                  |
| Java 15 |                   | Final       | 2nd Preview | 2nd Preview | Preview        |                  |
| Java 16 |                   |             | Final       | Final       | 2nd Preview    |                  |
| Java 17 |                   |             |             |             | Final          | Preview          |





# Pattern matching for switch

- Support for patterns in case labels
- Two additional patterns
- Guarded pattern
- Parenthesized pattern
- Special support for null values
- Switch pattern matching and sealed classes



# Pattern matching for switch

```
public class BasicPatternMatching {  
  
    public static void main(String... args) {  
        BasicPatternMatching app = new BasicPatternMatching();  
        System.out.println(app.printValue(obj: "12"));  
        System.out.println(app.printValue(obj: 42L));  
        System.out.println(app.printValue(obj: -30));  
        System.out.println(app.printValue(Boolean.TRUE));  
        System.out.println(app.printValue(obj: null));  
    }  
  
    private String printValue(Object obj) {  
        return switch (obj) {  
            case Integer i -> String.format("It is an integer with value %d", i);  
            case Long l -> String.format("It is a Long with value %d", l);  
            case String s -> String.format("It is a String with value %s", s);  
            case null -> new String(original: "You can't pass in a null value!");  
            default -> String.format("Dunno the type, but the value is %s", obj.toString());  
        };  
    }  
}
```



# Pattern matching for switch

```
public class GuardedPattern {  
  
    public static void main(String... args) {  
        GuardedPattern app = new GuardedPattern();  
        System.out.println(app.printValue( obj: "Hello"));  
        System.out.println(app.printValue( obj: "Hello World!"));  
    }  
  
    private String printValue(Object obj) {  
        return switch (obj) {  
            case null -> new String( original: "You can't pass in a null value!");  
            case String s && s.length() > 10 -> String.format("Long String with value %s", s);  
            case String s -> String.format("Not so long String with value %s", s);  
            default -> String.format("Dunno the type, but the value is %s", obj.toString());  
        };  
    }  
}
```



# Pattern matching for switch

```
public class ParenthesizedPatternMatching {  
  
    public static void main(String... args) {  
        ParenthesizedPatternMatching app = new ParenthesizedPatternMatching();  
        System.out.println(app.printValue(obj: "xx"));  
        System.out.println(app.printValue(obj: "!"));  
        System.out.println(app.printValue(obj: "@@"));  
    }  
  
    private boolean printValue(Object obj) {  
        return switch (obj) {  
            case String s && s.length() >= 2 && s.contains("@") || s.contains("!") -> true;  
            default -> false;  
        };  
    }  
}
```

false  
true  
true



# Pattern matching for switch

```
public class ParenthesizedPatternMatching {  
  
    public static void main(String... args) {  
        ParenthesizedPatternMatching app = new ParenthesizedPatternMatching();  
        System.out.println(app.printValue( obj: "xx"));  
        System.out.println(app.printValue( obj: "!"));  
        System.out.println(app.printValue( obj: "@@"));  
    }  
  
    private boolean printValue(Object obj) {  
        return switch (obj) {  
            case String s && s.length() >= 2 && (s.contains("@") || s.contains("!")) -> true;  
            default -> false;  
        };  
    }  
}
```

false  
false  
true



# Pattern matching for switch

```
public class SealedClassesPatternMatching {  
  
    public static void main(String... args) {  
        SealedClassesPatternMatching app = new SealedClassesPatternMatching();  
        Ocelot ocelot = new Ocelot();  
        System.out.println(app.printValue(ocelot));  
    }  
  
    private String printValue(Feline feline) {  
        return switch (feline) {  
            case Tiger t -> "It is a tiger";  
            case Puma p -> "It's a puma";  
            case Ocelot p -> "It's an ocelot";  
        };  
    }  
}
```



The background of the slide is a photograph of a large crowd at a concert at night. The crowd is silhouetted against bright stage lights, and many people have their hands raised in the air. The scene is filled with energy and excitement.

# [github.com/RonVeen/Java17Presentation](https://github.com/RonVeen/Java17Presentation)

The background of the slide is a photograph of a large crowd at a concert at night. The crowd is silhouetted against bright stage lights, and many people have their hands raised in the air. The text 'THANK YOU!' is superimposed in the center of the image in a large, white, bold, sans-serif font. On the left and right sides of the image, there are large screens displaying the text 'ALL I AM IS YOURS'.