Discovering Modern Java

What you need to know about Java 9, 10, 11, 12, 13 and beyond

Henri Tremblay
Managing Director, Head of TS Canada
TradingScreen
@henri_tremblay

Henri Tremblay

EASYMOCK

OBJENESIS

EHCACHE

- More or less made possible class mocking and proxying
- Coined the term “partial mocking”

Henri Tremblay

- More or less made possible class mocking and proxying
- Coined the term “partial mocking”

Good old

5

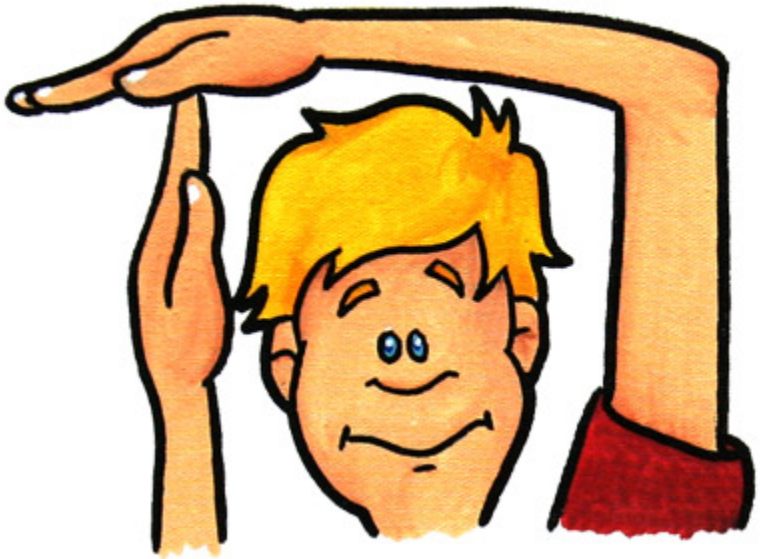
New & shinny

13

POLL

- Which version of Java are you currently using?
  - Java 6 or less
  - Java 7
  - Java 8
  - Java 9
  - Java 11

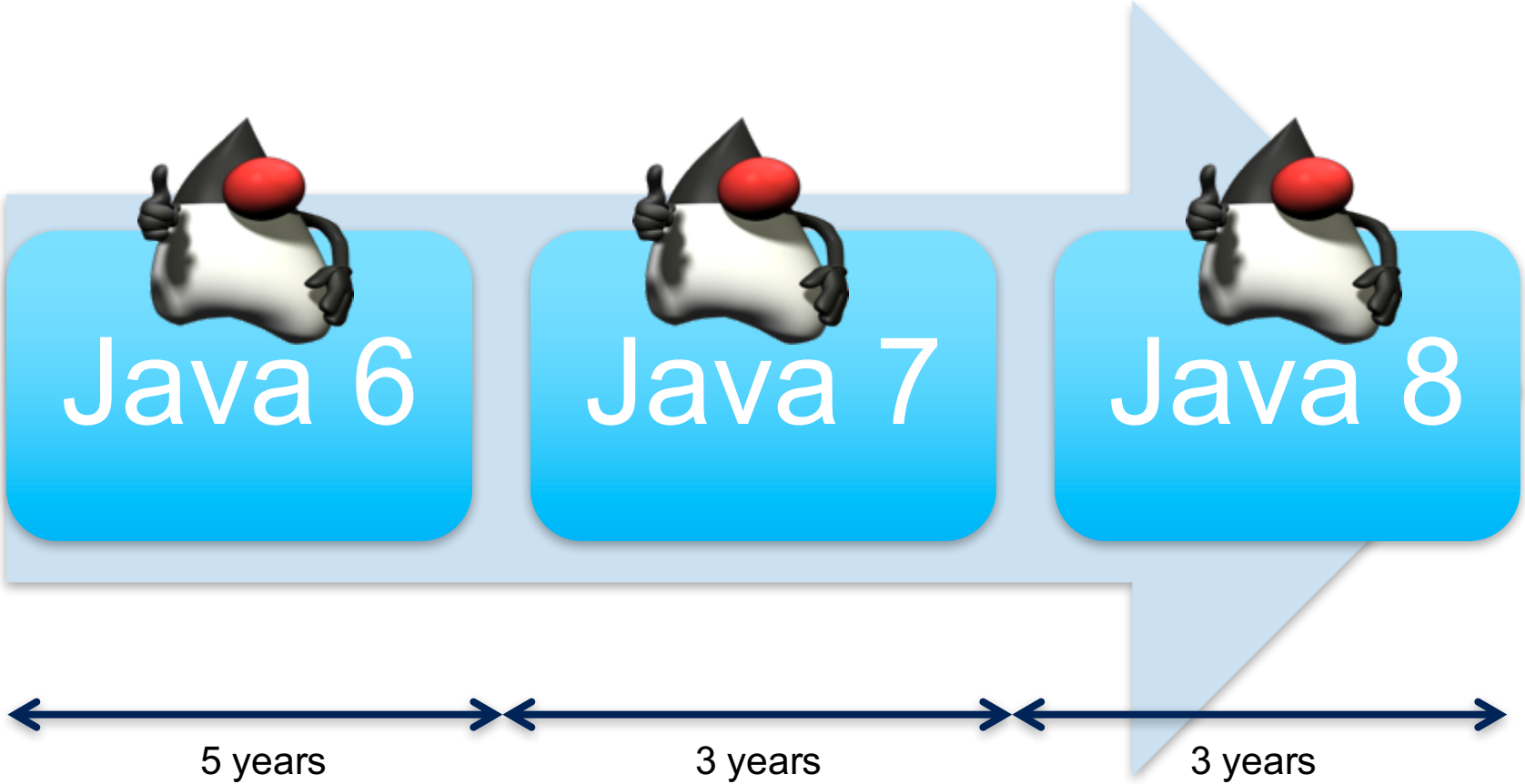
Breaks



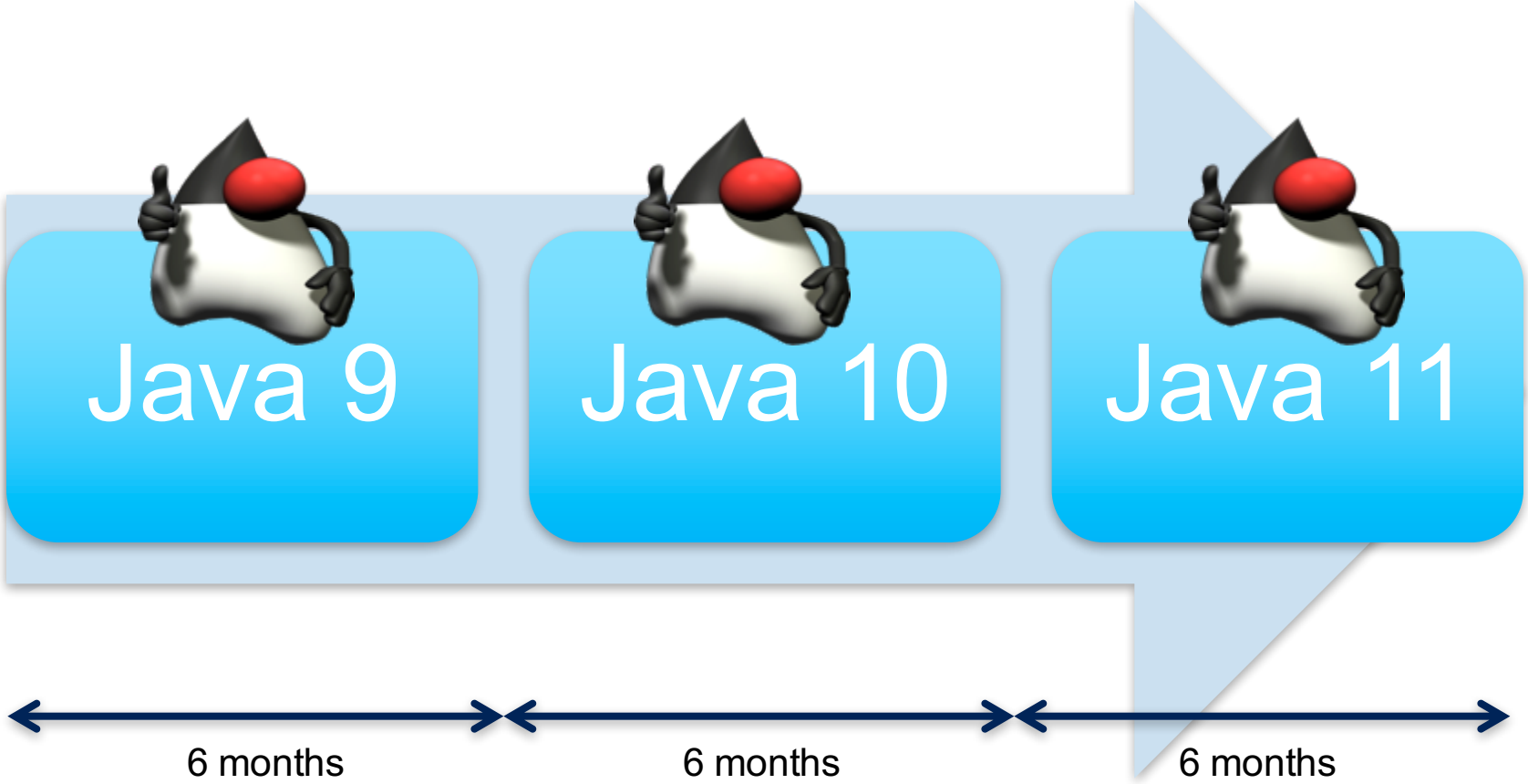
Questions

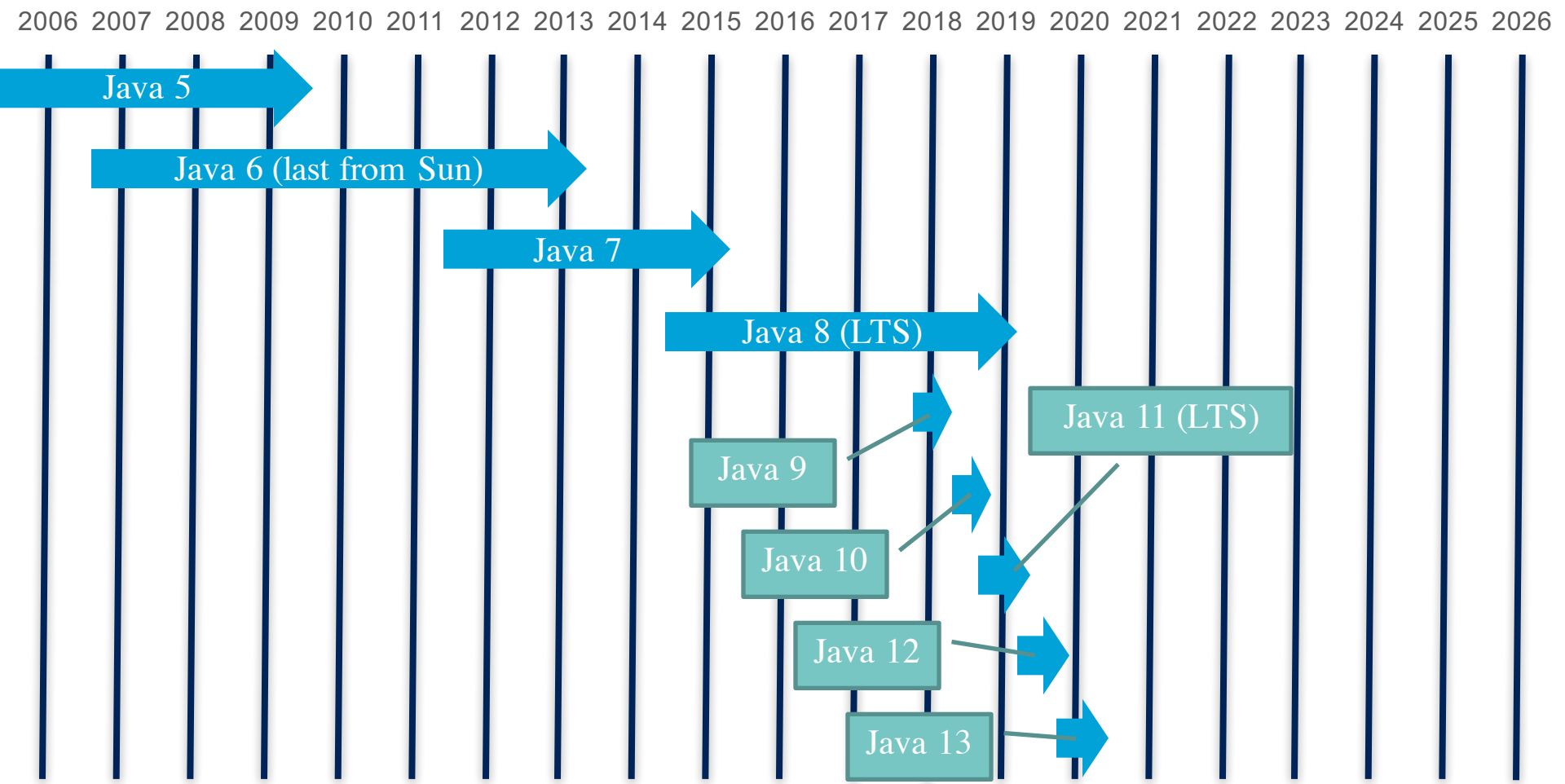
Java Delivery Process

Old delivery process



New delivery process




2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026
Java 5
Java 6 (last from Sun)
Java 7
Java 8 (LTS)
Java 9
Java 10
Java 11 (LTS)
Java 12
Java 13

TL;DR

- **OracleJDK**: Free for development, supported for 6 months
  - LTS versions have extended ($$$) support by Oracle
  - Identical to OpenJDK but with support
- **Oracle OpenJDK:** Free forever, supported for 6 months
- **Other OpenJDK:** Free forever, supported after 6 months by RedHat (Java 8 and 11)
  - Built by AdoptOpenJDK (https://adoptopenjdk.net/), Azul, IBM, Amazon…
- **Other JVMs** (Amazon, Azul, IBM, RedHat, etc.)
  - Supported by their vendor as they wish

- A lot of “May” and “Possibly” in the party line
  - https://medium.com/@javachampions/java-is-still-free-2-0-0-6b9aa8d6d244

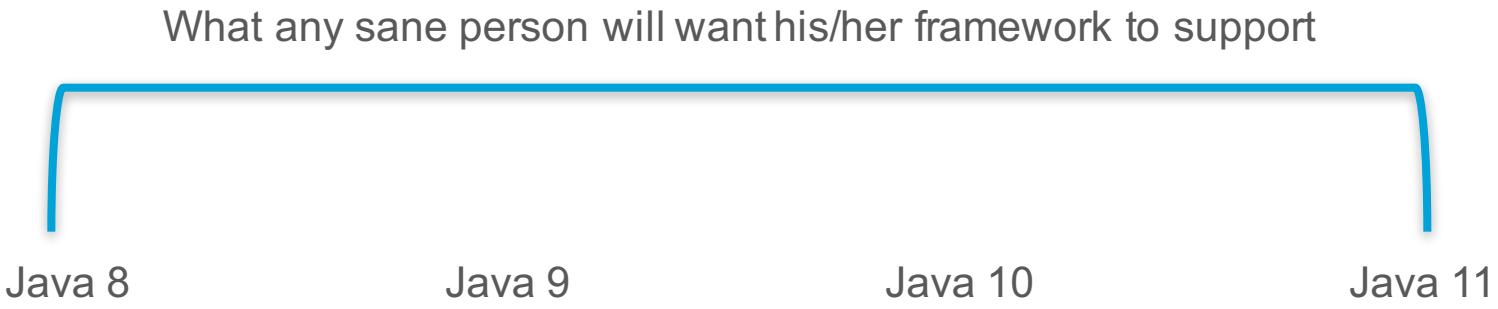
`@Deprecated`

is back

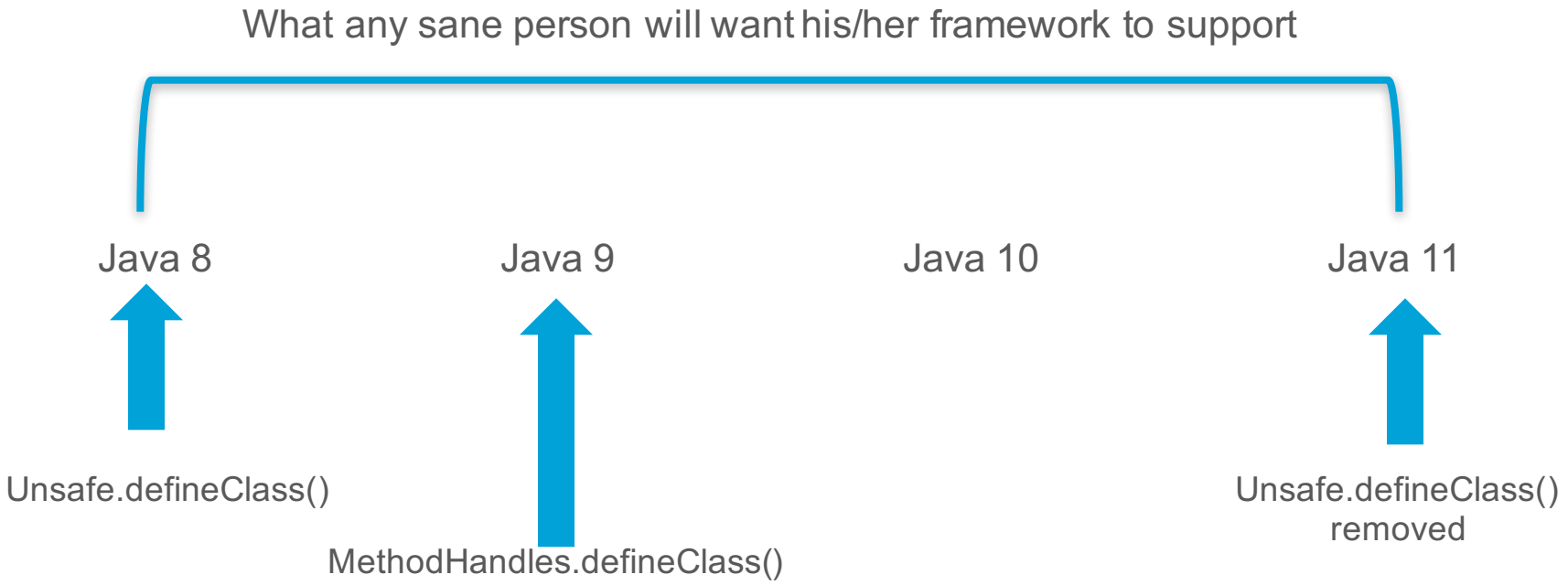
Stronger than ever

```
/**
 * Counts the number of stack frames in this thread. The thread must
 * be suspended.
 *
 * @return     the number of stack frames in this thread.
 * @throws     IllegalThreadStateException  if this thread is not
 *             suspended.
 * @deprecated The definition of this call depends on {@link #suspend},
 *             which is deprecated.  Further, the results of this call
 *             were never well-defined.
 *             This method is subject to removal in a future version of Java SE.
 * @see        StackWalker
 */
@Deprecated(since="1.2", forRemoval=true)
public native int countStackFrames();
```

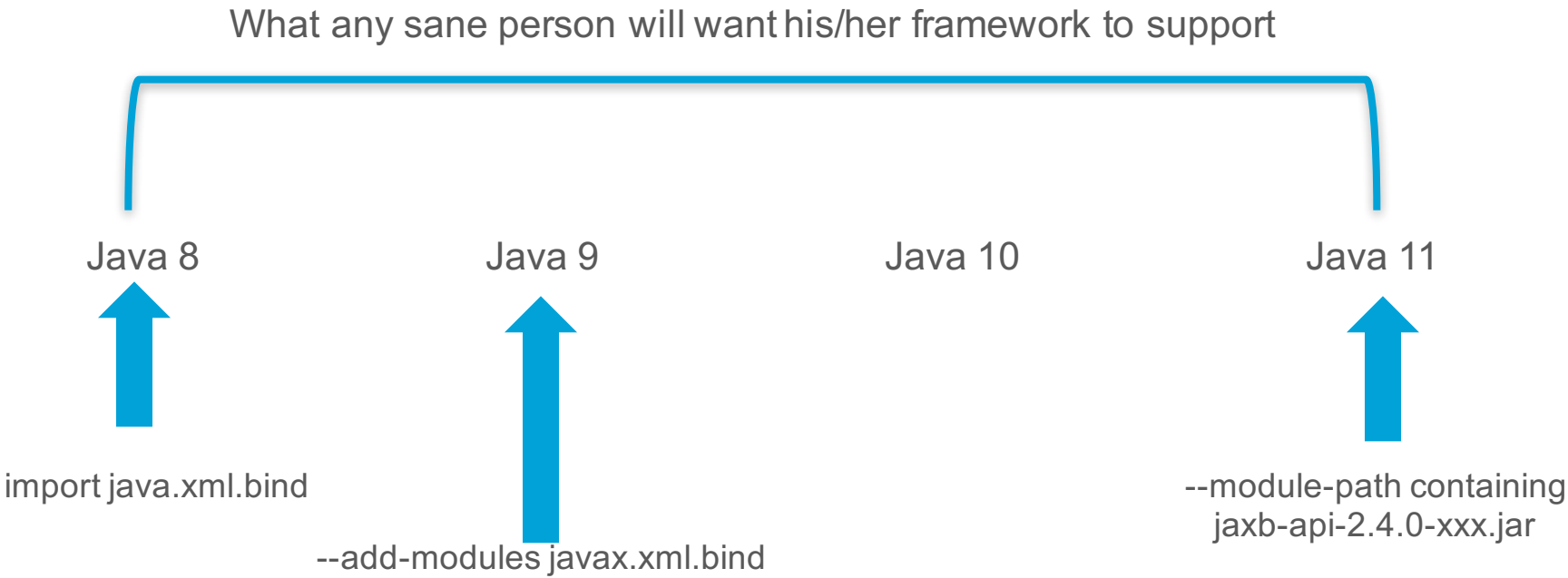
Prepare to do some stretching



Prepare to do some stretching



Prepare to do some stretching



GC

Garbage collectors for all

Java < 8

- Serial
- Parallel
- ParallelOld
- CMS
- iCMS

Java 8

- G1

Java 9

- Removed iCMS

Java 11

- Epsilon
- Z

Java 12

- Shenandoah

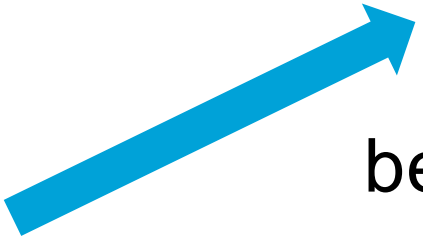
Java 5

(2004)

JAVA 5 GAVE THE GENERIC TYPES TO THE WORLD

(and also annotations, concurrent collections, enum types, for each, static imports and so on and so on)

```
MyClass.<List<String>> anyObject()
```



because you can’t

```
(List<String>) MyClass.anyObject()
```

```
MyClass.<List<String>> anyObject()
```

because you can’t

```
(List<String>) MyClass.anyObject()
```

DEMO

Java 6

(2006, last from Sun)

JAVA 6 BROUGHT.. PRETTY MUCH NOTHING

(a bunch of performance improvements under the hood, better xml parsing and the first scripting api)

Java 7

(2011, first from Oracle)

JAVA 7 BROUGHT A LOT OF SYNTACTIC SUGAR

(plus invokeDynamic, forkJoin, better file IO)

```
switch(s) {
    case "hello":
        return "world";
    case "bonjour":
        return "le monde";
}
```

DEMO

```
List<String> list = new ArrayList<>();
```

Binary integer literals and underscores

```
int i = 0b1110001111;

int i = 1_000_000;
```

Multiple catches

```
try {
    // ... do stuff
}
catch (IOException | SerializationException e) {
    log.error("My error", e);
}
```

Instead of

```
try {
    // ... do stuff
} catch (IOException e) {
    log.error("My error", e);
} catch (SerializationException e) {
    log.error("My error", e);
}
```

Before

```
InputStream in = new FileInputStream("allo.txt");
try {
   // … do stuff
} finally {
   try { in.close(); } catch(IOException e) {}
}
```

After

```
try(InputStream in = new FileInputStream("allo.txt")) {
   // … do stuff
}
```

Real code you should do

```
InputStream in = new FileInputStream("allo.txt");
try {
   // … do stuff
   in.close();
} catch(IOException e) {
   try {
      in.close();
   } catch(IOException e1) {
      e.addSuppressed(e1);
   }
   throw e;
}
```

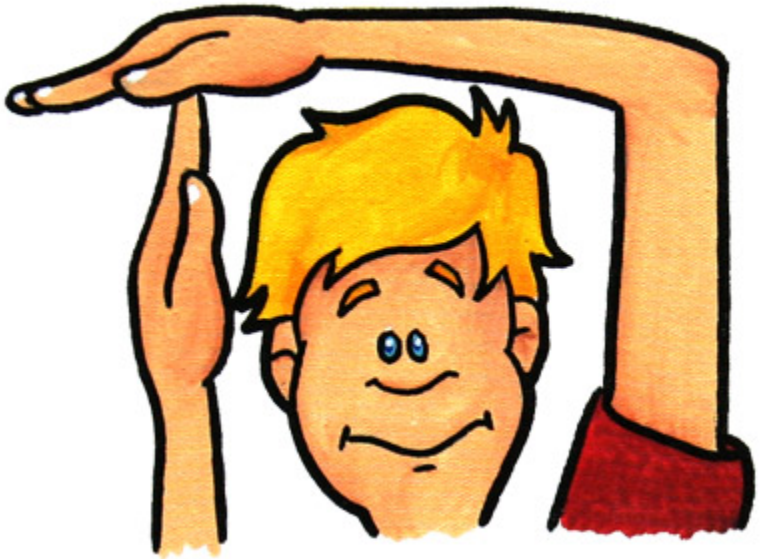
```
List<String> lines =
    Files.readAllLines(
        Paths.get("path", "to", "my", "file.txt"));
```

// also: file watcher, symbolic links, file locks,
// copy … Look in `java.nio.file`

DEMO

DEMO

5 minutes break



Java 8

(2014)

JAVA 8: LAMBDA!

(and also a new date API, default methods, metaspace, Nashorn, JavaFX and CompletableFuture)

Base64 ;-)

```
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class Base64s {
    public static void main(String[] args) {
        final String text = "Base64 finally in Java 8!";

        final String encoded = Base64
                .getEncoder()
                .encodeToString( text.getBytes( StandardCharsets.UTF_8 ) );
        System.out.println( encoded );

        final String decoded = new String(
                Base64.getDecoder().decode( encoded ),
                StandardCharsets.UTF_8 );
        System.out.println( decoded );
    }
}
```

- Core ideas:
  - Immutable
  - A time is a time, a date is a date. Not always both (like `java.util.Date`)
  - Not everyone uses the same calendar (the same Chronology)
- `LocalDate`, `LocalTime`, `LocalDateTime` → Local. No time zone
- `OffsetDateTime`, `OffsetTime` → Time with an offset from Greenwich
- `ZonedDateTime` → `LocalDateTime` with a time zone
- `Duration`, `Period` → Time span
- `Instant` → Timestamp
- `Formatting` → Easy and thread-safe formatting

```
LocalDateTime now = LocalDateTime.now();
String thatSpecialDay = now
    .withDayOfMonth(1)
    .atZone(ZoneId.of("Europe/Paris"))
    .plus(Duration.ofDays(5))
    .format(DateTimeFormatter.ISO_ZONED_DATE_TIME);

System.out.println(thatSpecialDay);
```

Output

2016-09-06T17:45:22.488+01:00[Europe/Paris]

Lambda: 11th letter of the Greek alphabet

(also written as λ-calculus) is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution. It is a universal model of computation that can be used to simulate any single-taped Turing machine and was first introduced by mathematician Alonzo Church in the 1930s as part of an investigation into the foundations of mathematics.

This is a function:

$$\mathrm{square\_sum}(x, y) = x^2 + y^2$$

This is a lambda:

$$(x, y) \mapsto x^2 + y^2$$

```
int value = numbers.stream()
  .reduce(0, (a, b) -> pow(a, 2) + pow(b, 2));
```

```
// One-liner, one arg
list.stream()
  .map(name -> t(name))
  .forEach(name -> println(name));

// Multiple arguments
map.forEach((key, value) ->
  println(key + "=" + value));

// Typed
list.forEach((String e) -> println(e));
```

```
// Multiline
list.stream()
  .map(name -> {
    String result = t(name);
    return result;
  })
  .forEach(name -> {
    println(name);
  });

// With closure
String greeting= "Hello ";
list.forEach(e -> println(greeting + e));
```

DEMO

```
List<String> list = new ArrayList<>();
String greeting = "Hello "; // no final required

list.forEach(s -> System.out.println(greeting + s));

list.forEach(new Consumer<String>() {
    @Override public void accept(String s) {
        System.out.println(greeting + s);
    }
});

list.forEach(s -> greeting = “Hi”); // won’t compile
```

DEMO

Interface default methods

```
public interface List<E> extends Collection<E> {
  default void replaceAll(UnaryOperator<E> operator)
{
    // …
  }
}

public interface A {
    default void foo() { }
}

public interface B{
    default void foo() { }
}
```

```
public class C implements A, B {} // forbidden

public class C implements A, B { // allowed
    public void foo() { }
}

public static class D implements A, B { // allowed
    public void foo() {
        A.super.foo();
    }
}
```

```
public interface IntStream {
    static IntStream empty () {
        return …;
    }
}
```

```
IntStream stream = IntStream.empty ();
```

DEMO

Method references

```
public static class Passenger {
    public void inboard(Train train) {
        System.out.println("Inboard " + train);
    }
}

public static class Train {
    public static Train create(Supplier< Train >
supplier) {
        return supplier.get();
    }

    public static void paintBlue(Train train) {
        System.out.println("Painted blue " + train);
    }

    public void repair() {
        System.out.println( "Repaired " + this);
    }
}
```

```
List<Train> trains = Arrays.asList(train);

 // static method
trains.forEach(Train::paintBlue);

 // instance method
trains.forEach(Train::repair);

// instance method taking this in param
Passenger p = new Passenger();
trains.forEach(p::inboard);

trains.forEach(System.out::println);      // useful!

 // constructor
Train train = Train.create(Train::new);
```

NullPointerException on method reference 08

DEMO

DEMO

```
List<String> list = new ArrayList<>();
int sum = list
        .parallelStream()
        .filter(s -> s.startsWith("a"))
        .mapToInt(String::length)
        .sum();


Arrays.parallelSort(array);
```

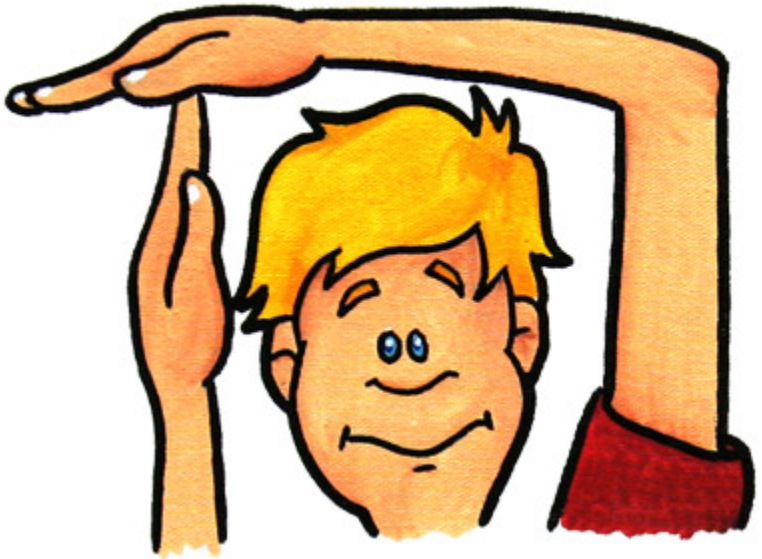
DEMO

- Do not pass it in parameter
  - `public void print(Optional<String> message)`

- Returning it is fine
  - But remember, it adds to object allocation
- Don’t forget primitive types
  - OptionalInt, OptionalLong, OptionalDouble

DEMO

5 minutes break



Java 9

(2017)

JAVA 9: MODULES!

(and G1 by default, var handles, new http client, jshell, jcmd, immutable collections, unified JVM logging)

```
Map<String, String> map = new HashMap<>() {{
    put("key", "value");
}};
```


```
Map<String, String> map =
    Map.of("key", "value");
```

Slightly longer map instantiation

```
private Map<String, String> map = new HashMap<>();
{
  map.put("key", "value");
}
```

Improved try-with-resource

```
ByteArrayOutputStream in = new ByteArrayOutputStream();

try(in) {

}
```

```
public interface Printer {
  private void print(String s) {
    System.out.println(s);
  }

  default void printAll(String[] list) {
    Arrays.stream(list).forEach(this::print);
  }
}
```

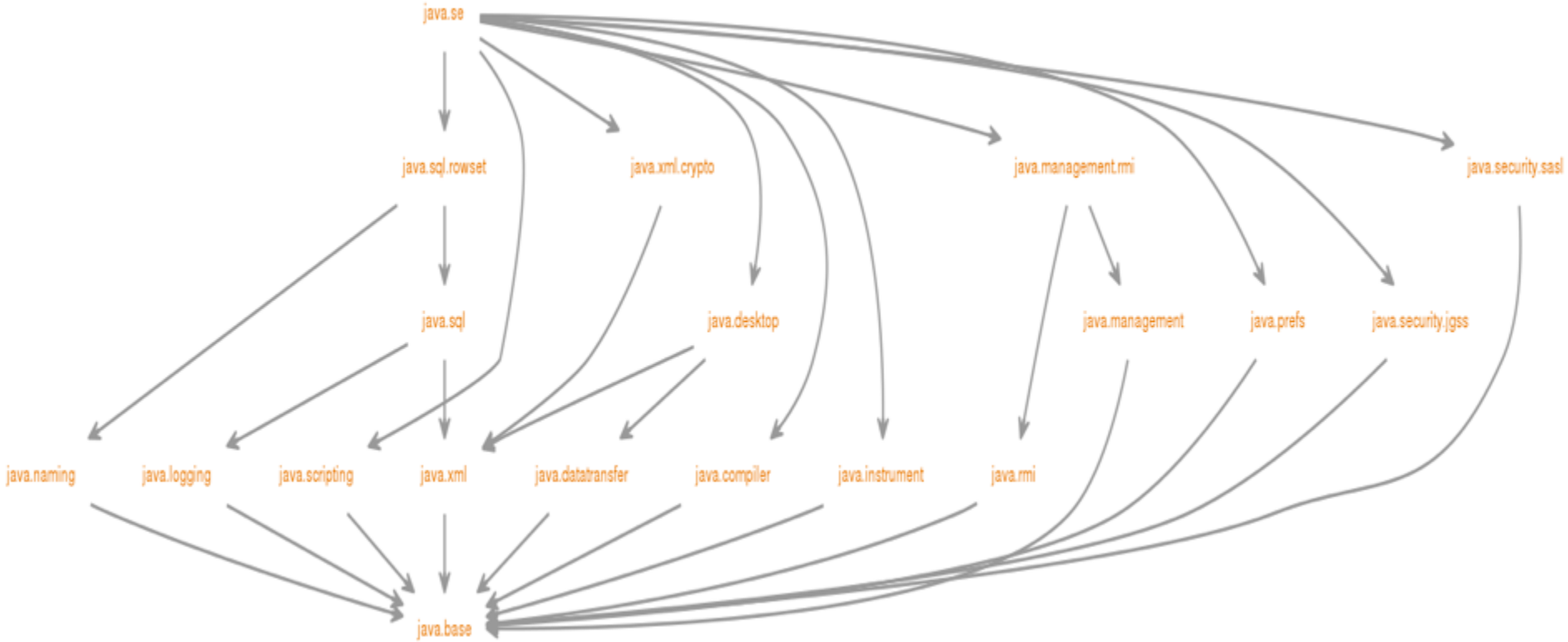
DEMO

```
MethodHandle mh = MethodHandles.lookup()
  .findSetter(getClass(), "name", String.class);
mh.invokeExact(this, ”Henri");

// or

VarHandle mh = MethodHandles.lookup()
  .findVarHandle(getClass(), "name", String.class);
mh.set(this, ”Henri");
```

Java SE Modules



Credits: Oracle

DEMO

Java 10

(March 2018)

JAVA 10: VAR

(application class-data sharing, GraalVM)

- Keywords
  - `while, if, abstract, public, default, class, enum (Java 5), _ (Java 9)`
- Restricted Keywords
  - open, module, to, with
- Literals
  - true, false, null
- Reserved identifier
  - var

DEMO

Java 11

(September 2018)

JAVA 11: CLEANUP

(and dynamic class-file constants, epsilon, java.xml, corba and many J2EE modules removed, nashorn deprecated, ZGC, single file program)

DEMO

Java 12

(March 2019)

JAVA 12: SWITCH EXPRESSIONS

Switch expressions

```
private boolean isWeekDay(DayOfWeek day) {
  boolean weekDay;

  switch(day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
    case THURSDAY:
    case FRIDAY:
      weekDay = true;
      break;
    case SATURDAY:
    case SUNDAY:
      weekDay = false;
    default:
      throw new IllegalStateException("A new day was added in my week: " + day);
  }
  return weekDay;
}
```

Switch expressions

```
private boolean isWeekDay(DayOfWeek day) {
  boolean weekDay;

  switch(day) {
    case MONDAY:
    case TUESDAY:
    case WEDNESDAY:
    case THURSDAY:
    case FRIDAY:
      weekDay = true;
      break;
    case SATURDAY:
    case SUNDAY:
      weekDay = false;
      break;
    default:
      throw new IllegalStateException("A new day was added in my week: " + day);
  }
  return weekDay;
}
```

Switch expressions

```
private boolean isWeekDay(DayOfWeek day) {
  boolean weekDay;

  switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> weekDay = true;
    case SATURDAY, SUNDAY -> weekDay = false;
    default -> throw new IllegalStateException("A new day was added in my week: " + day);
  }

  return weekDay;
}
```

Switch expressions

```
private boolean isWeekDay(DayOfWeek day) {
  boolean weekDay = switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> true;
    case SATURDAY, SUNDAY -> false;
    default -> throw new IllegalStateException("A new day was added in my week: " + day);
  };

  return weekDay;
}
```

Switch expressions

```
private boolean isWeekDay(DayOfWeek day) {
  return switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> true;
    case SATURDAY, SUNDAY -> false;
    default -> throw new IllegalStateException("A new day was added in my week: " + day);
  };
}
```

Java 13

(September 2019)

JAVA 13: TEXT BLOCK!

(and also more switch expressions, new socket API implementation and ZGC soft heap)

Switch expressions (still a preview)

```
private boolean isWeekDay(DayOfWeek day) {
  return switch(day) {
    case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> true;
    case SATURDAY, SUNDAY -> false;
    default -> {
        if(onEarth) {
            yield false;
        }
        yield true;
    }
  };
}
```

```
String script = """
                function hello() {
                  print('"Hello, world\"');
                }

                hello();
                """;
```

Conclusion

Who has learned something today?

Links

- Java Champions – Java is still free
  - https://medium.com/@javachampions/java-is-still-free-2-0-0-6b9aa8d6d244
- Adopt OpenJDK
  - https://adoptopenjdk.net/
- Maurice Naftalin's Lambda FAQ
  - http://www.lambdafaq.org/
- Zeroturnaround Module Cheat Sheet
  - http://files.zeroturnaround.com/pdf/RebelLabs-Java-9-modules-cheat-sheet.pdf
- Var styleguide
  - http://openjdk.java.net/projects/amber/LVTIstyle.html

- Mastering Lambdas
  - Maurice Naftalin
- Java 9 Modularity
  - Sander Mak and Paul Bakker
- Courses by Cay S. Horstmann on Safari

http://objenesis.org

http://montreal-jug.org

Questions?

Henri Tremblay
henri@tremblay.pro
http://blog.tremblay.pro
@henri_tremblay

EASYMOCK

http://easymock.org