From Java 17 to 21 and beyond: Amber – Loom – Valhalla

José Paumard
Java Developer Advocate
Java Platform Group

https://twitter.com/JosePaumard

https://github.com/JosePaumard

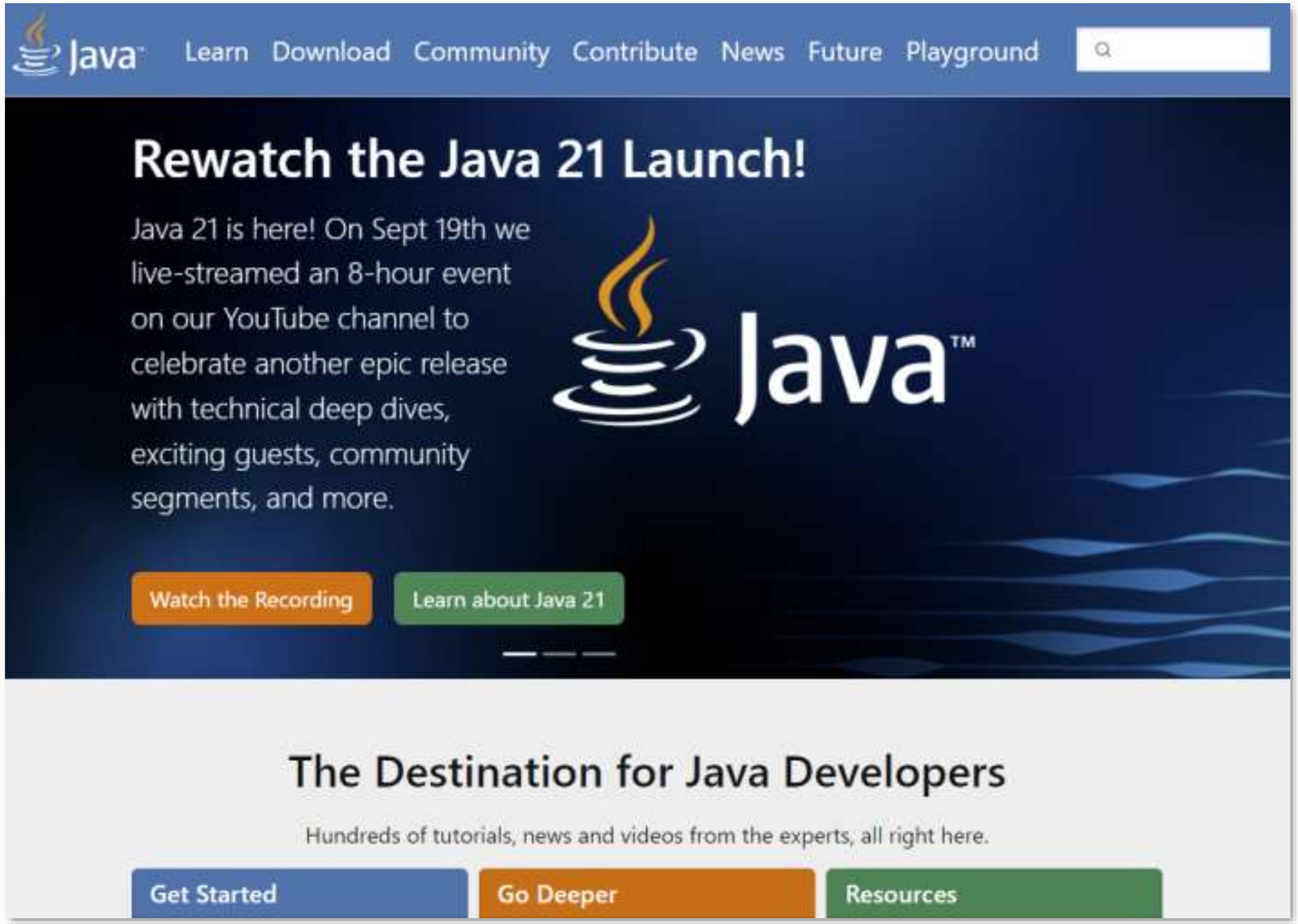
https://www.youtube.com/c/JosePaumard01
https://www.youtube.com/user/java
https://www.youtube.com/hashtag/jepcafe

https://fr.slideshare.net/jpaumard

https://www.pluralsight.com/authors/jose-paumard

https://dev.java

https://dev.java/



Rewatch the Java 21 Launch!

Java 21 is here! On Sept 19th we live-streamed an 8-hour event on our YouTube channel to celebrate another epic release with technical deep dives, exciting guests, community segments, and more.

The Destination for Java Developers

Hundreds of tutorials, news and videos from the experts, all right here.

Tune in!

How to Upgrade to Java 21 #RoadTo21

JDK 21 Release Notes - Inside Java Newscast #55

Inside Java Newscast

Java 21 JVM & GC Improvements #RoadTo21

Java 21 Tool Enhancements: Better Across the Board...

Road To 21 series

Inside Java Podcast

```
var ordered = List.of(1, 2, 3);
var reversedOrdered = list.reversed();

var sorted = new TreeSet<>(list);
var reversedSorted = sorted.reversed();

var unmodifiable = Collections.
    unmodifiableSequencedSet(sorted);
```

Java 21 New Feature: Sequenced Collections - JE...

JEP Café

```
Runnable runnable =
  () -> { doSomething(); };

Callable<Integer> callable =
  () -> { return anInt(); };
```

Cracking the Java coding interview

Inside Java
News and views from members of the Java team at Oracle

Shows: Podcast | Newscast | JEP Café | Sip Of Java — dev.java | About | Jobs

Sort by: Date | Author | Tag

Java 20 Unboxing - Inside Java Newscast #44
Billy Korando on March 21, 2023

Java First. Java Always. | Level Up Keynote
Georges Saab on March 22, 2023

JDK 20 Security Enhancements
Sean Mullan on March 22, 2023

The Arrival of Java 20!
Sharat Chander on March 21, 2023

Inside.java

Java 21 in Two Minutes... more or less

Sip of Java

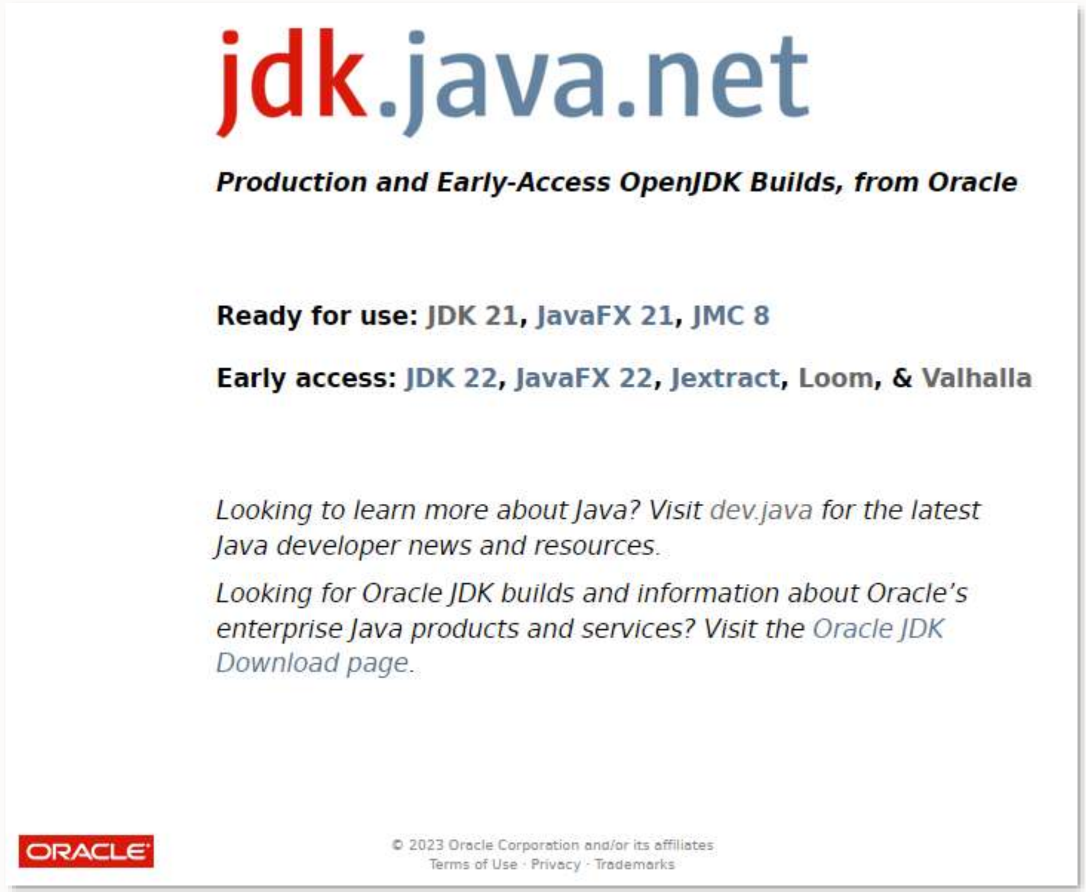
OpenJDK is the place where it all happens

OpenJDK

What is this? The place to collaborate on an open-source implementation of the Java Platform, Standard Edition, and related projects.

https://openjdk.org/

OpenJDK is the place where it all happens



jdk.java.net

Production and Early-Access OpenJDK Builds, from Oracle

Ready for use: JDK 21, JavaFX 21, JMC 8

Early access: JDK 22, JavaFX 22, Jextract, Loom, & Valhalla

Looking to learn more about Java? Visit dev.java for the latest Java developer news and resources.

Looking for Oracle JDK builds and information about Oracle's enterprise Java products and services? Visit the Oracle JDK Download page.

© 2023 Oracle Corporation and/or its affiliates
Terms of Use · Privacy · Trademarks

https://jdk.java.net/

Amber, Loom, Valhalla

Amber, Loom, Valhalla

Amber

- adding Pattern Matching to the Java language
- adding small language features to enhance productivity

Data Oriented Programming!

Amber, Loom, Valhalla

Loom

- bring a new concurrent programming model
- add virtual threads
- structured concurrency
- and scoped values

Get rid of Reactive Programming!

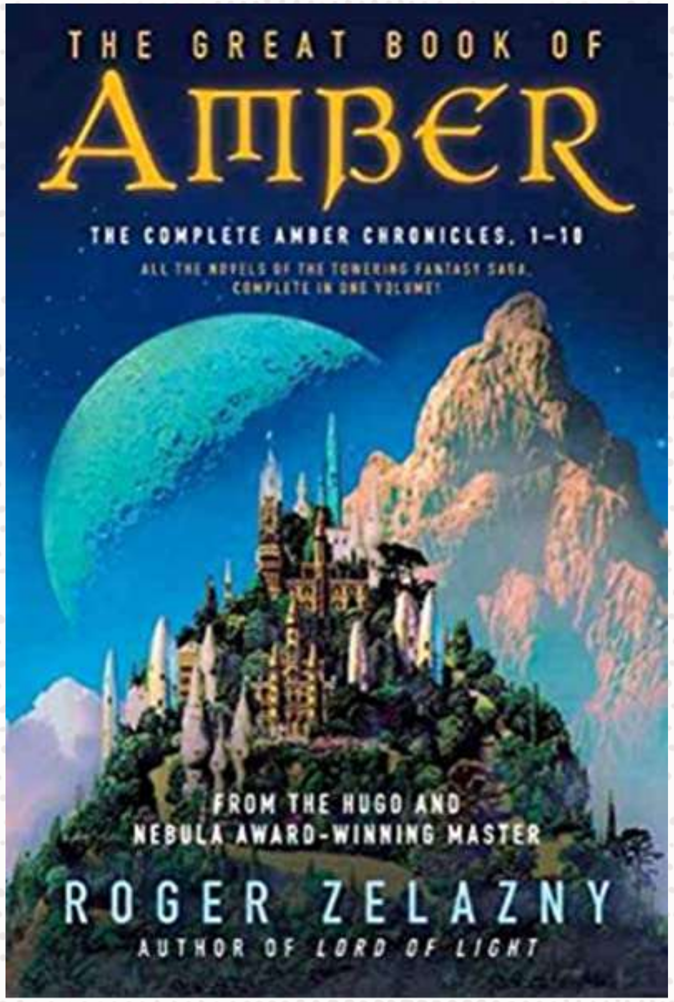
Amber, Loom, Valhalla

Valhalla

- value types
- user defined primitive types
- specialized generics

Do not choose between a clear model and performances!

Project Amber


THE GREAT BOOK OF
AMBER
THE COMPLETE AMBER CHRONICLES, 1–10
ALL THE NOVELS OF THE TOWERING FANTASY SAGA, COMPLETE IN ONE VOLUME!
FROM THE HUGO AND NEBULA AWARD-WINNING MASTER
ROGER ZELAZNY
AUTHOR OF LORD OF LIGHT

OOP according to Java

Encapsulation

```
class City {
   private String name;

   public String name() {
      this.name
   }
}
```

OOP according to Java

Interface and sub-typing

```
interface Populated {
   int population();
}
```

```
class City implements Populated { ... }
```

```
Populated populated = new City(...);
```

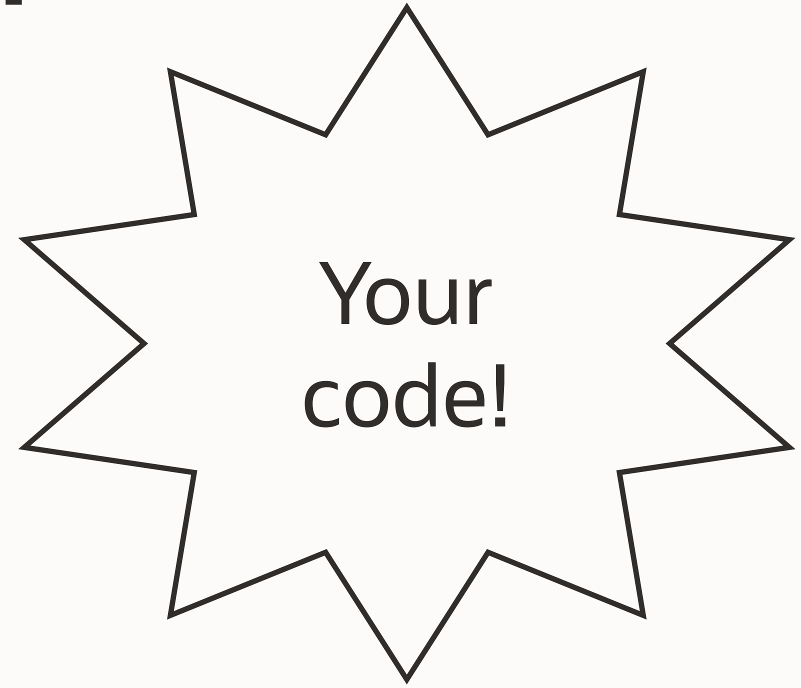
OOP according to Java

Late binding (virtual call or polymorphism)

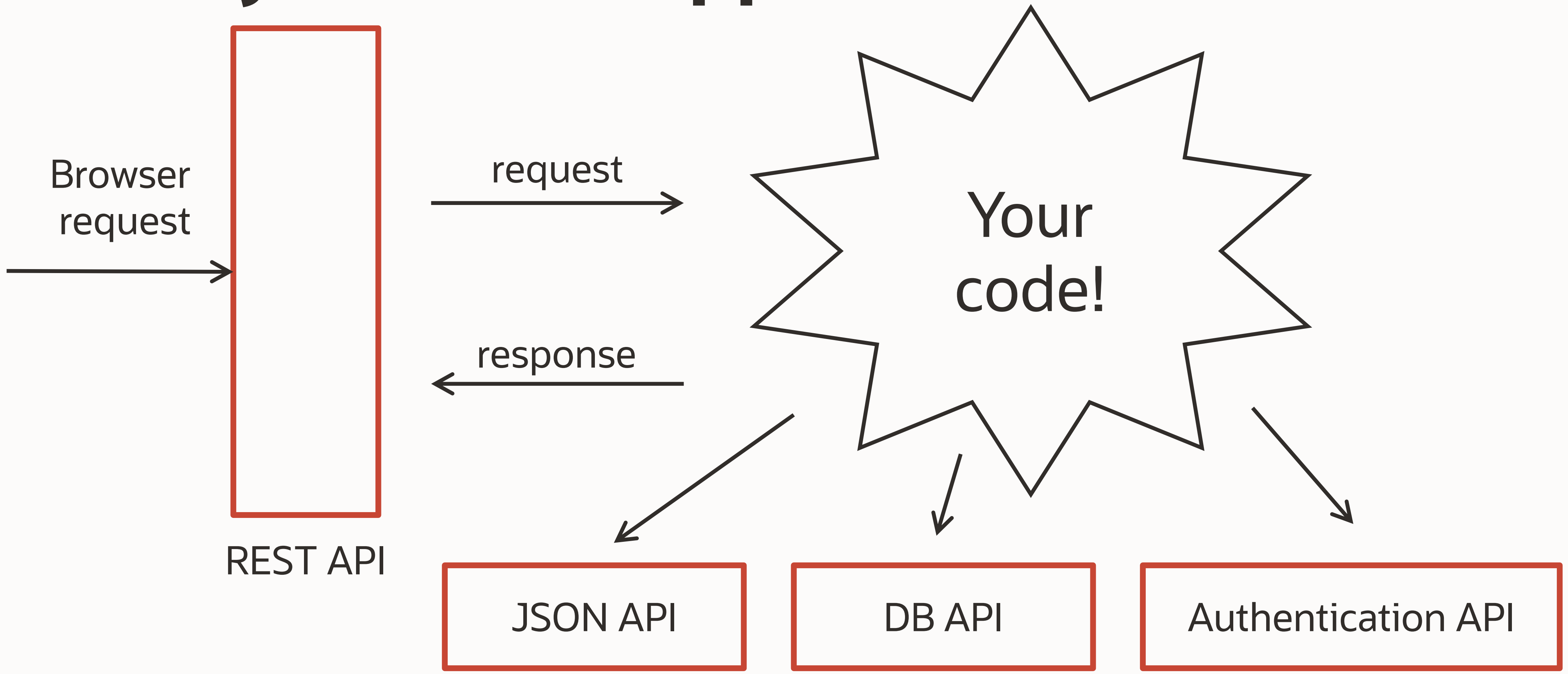
```java
interface Populated {
   int population();
}
```

```java
Populated populated = new City(...);
var population = populated.population();
```

Anatomy of a Web Application



Anatomy of a Web Application



SUCCESS STORY!

OOP in Java

Interfaces are driving the way you organize your applications

DOP in Java

Data is driving your code

Data First!

Object Oriented Programming

Model the problem using a class and an interface

+ late binding

```
final class City implements Populated {
private final int population;

   public int population() {
      return population;
   }
}
```

```
interface Populated {
   public int population();
}
```

```
final class Department implements Populated {
   private final String population;

   public int population() {
      return population;
   }
}
```

Consequences

When an interface changes, the compiler tells you what classes need to be updated, which is great!

Drawbacks?

Everytime a new businesss requirement shows up, you end up adding methods in your interfaces

Soon, you will have many fields and many methods in your Object Model classes

Drawbacks!

1) Your business modules depend on these classes, but they depend on elements they don’t use!

2) Because every module depend on the Object Model, changing it becomes more and more expensive

3) Do you remove some code that is not used anymore?

Data Oriented Programming

Separate Data and Code

```
interface Populated { }
```

```
final class City
implements Populated { }
```

```
final class Department
implements Populated { }
```

Data Oriented Programming

Separate Data and Code

```
interface Populated { }
```

```
final class City
implements Populated { }
```

```
final class Department
implements Populated { }
```

```
static String population(Populated populated) {
    if (populated instanceof City) {
        var city = (City) populated;
        return city.name();
    }
    if (populated instanceof Department) {
        var department = (Department) populated;
        return department.name();
    }
    throw new AssertionError();
}
```

Data Oriented Programming

Separate Data and Code

```
interface Populated { }
```

```
final class City
implements Populated { }
```

```
final class Department
implements Populated { }
```

```
static String population(Populated populated) {
   if (populated instanceof City) {
      var city = (City) populated;
      return city.population();
   }
   if (populated instanceof Department) {
      var department = (Department) populated;
      return department.population();
   }
   throw new AssertionError();
}
```

Problems?

Your compiler cannot help you anymore

You need three language features for the compiler to be able to help you again

- Records and Sealed Type to model your data. If you change your model, the compiler can help you
- Pattern Matching: to deconstruct your records
- A new switch on sealed types

Data Oriented Programming

Separate Data and Code

```
sealed interface Populated
permits City, Department { }
```

```
record City(String name, int population)
implements Populated { }
```

```
record Department(String name, int population)
implements Populated { }
```

Data Oriented Programming

```java
static String population(Populated populated) {
   if (populated instanceof City) {
      var city = (City) populated;
      return city.population();
   }
   if (populated instanceof Department) {
      var department = (Department) populated;
      return department.population();
   }
   throw new AssertionError();
}
```

Data Oriented Programming

```java
static String population(Populated populated) {
   if (populated instanceof City) {
      var city = (City) populated;
      return city.population();
   }
   if (populated instanceof Department) {
      var department = (Department) populated;
      return department.population();
   }
   throw new AssertionError();
}
```

```java
static String population(Populated populated) {

   return switch(populated) {
      case City(String _, int population) -> population;
      case Department(String _, int population) -> population;
   };
}
```

Data Oriented Programming

```
static String population(Populated populated) {
   if (populated instanceof City) {
      var city = (City) populated;
      return city.population();
   }
   if (populated instanceof Department) {
      var department = (Department) populated;
      return department.population();
   }
   throw new AssertionError();
}
```

```
static String population(Populated populated) {

   return switch(populated) {
      case City(String _, int population) -> population;
      case Department(String _, int population) -> population;
   };
}
```

Amber: Pattern Matching

Pattern matching for instanceof (type pattern)

```
if (o instanceof User user) {

   String name = user.getName();

   // my business code
}
```

Amber: Record Pattern

Pattern matching for instanceof (record pattern)

```
record User(String name, int age) { }
```

```
if (o instanceof User(String name, int age)) {

   // use name and age
}
```

The Unnamed Pattern (prev 21)

The unnamed pattern avoids the calling of an accessor when it is not needed

```
record Point(double x, double y) {}
record Circle(Point center, double radius) {}
```

```
if (o instance Circle(_, var radius)) {
   var surface = PI*radius*radius;
   ...
}
```

Amber: Switch Expression + Record Pattern

```
sealed interface Shape
permits Square, Circle { }
```

```
int surface = switch (shape) {
   case Square(int edge) -> edge*edge;
   case Circle(int radius) -> PI*radius*radius;
};
```

Data Oriented Programming

Data is more important than code

- Not always true

The compiler can help you (like with OOP)

- Records define your data
- Sealed types make switch exhaustive
- Record patterns detect structural modification

Wadler's Expression of the Problem

If you are not the owner of the code

OOP - Polymorphism
- Add new subtypes
- No new operations

DOP
- Add new operations
- No new subtypes

You cannot get both ☹

Phil Wadler

More Patterns: Record Pattern on Classes

Deconstructor enables record pattern on class

```
class Point {

   private int x, y;

   matcher(int x, int y) Point { // provisional syntax
      match this.x, this.y;
   }
}
```

Named Pattern

Allows matcher methods to be named

```
Optional<String> opt = ...;

switch(opt) {
   case Optional.of(String s) -> ...;
   case Optional.empty() ->
}
```

```
final class Optional<T> {

   final private T value;
   final private boolean present;

   matcher(T t) of { // provisional syntax
      if (present) match this.value;
      no-match;
   }

   matcher() empty { // provisional syntax
      if (!present) match;
      no-match;
   }
}
```

Imperative Destructuring

Using record pattern in assignments

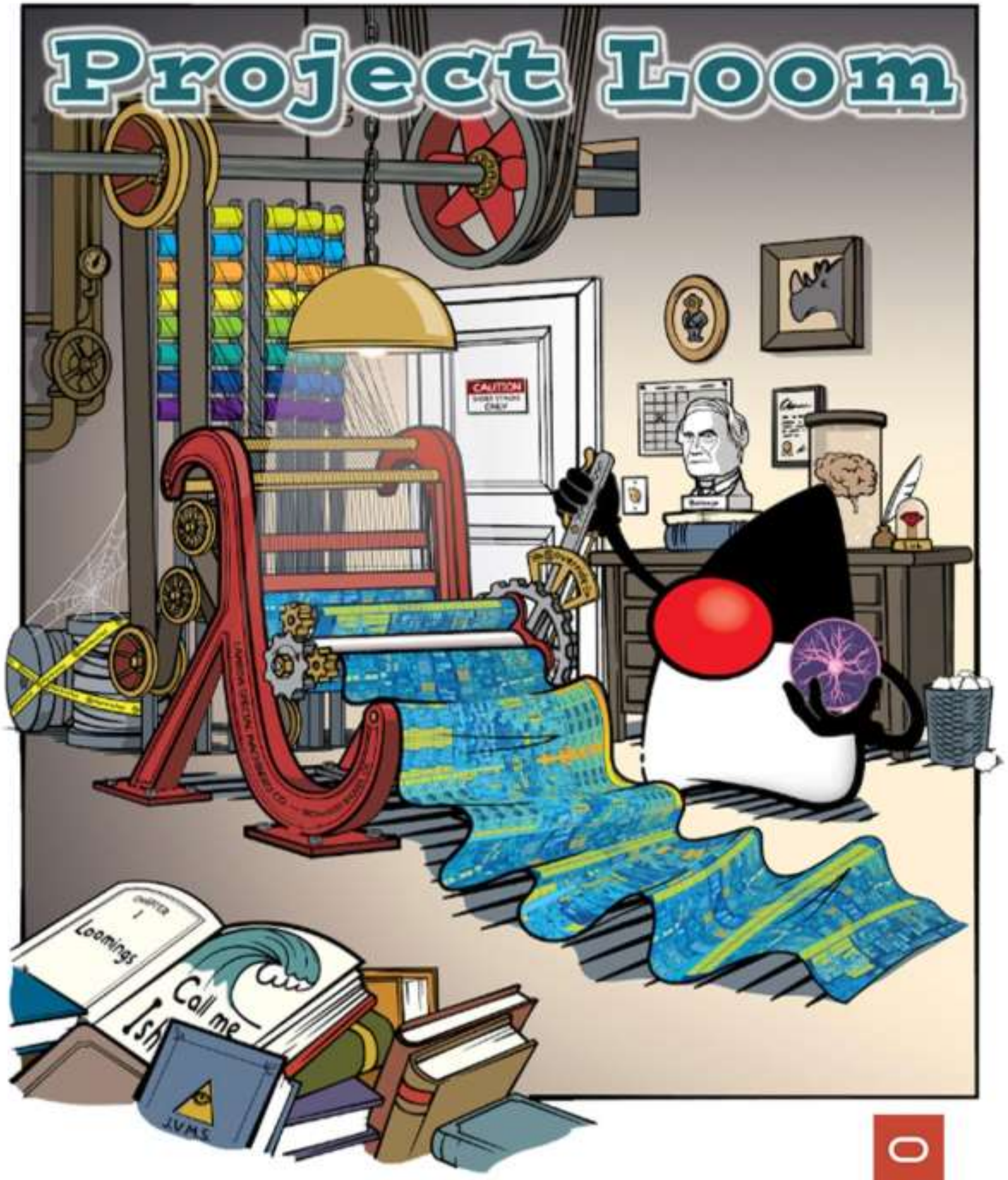
```
Point p = ...;
let Point(int x, int y) = p;
```

```
for (let Map.Entry(var key, var value): entrySet) {
    ...
}
```

Amber: Current Plan for the JDK 21

	JDK 21	JDK 22
Type Pattern	Final	
Record Pattern	Final	
Unamed Pattern	Preview	?
Named Pattern		?
Assignement		?

Loom



Loom: Virtual Threads

What does Loom want to fix?

Reactive programming!

Concurrency Issues

What is wrong with this code?

```
ExecutorService es = ...;
var f1 = es.submit(SomeService::readImages);
var f2 = es.submit(SomeService::readLinks);

Page page = new Page(f1.get(1, TimeUnit.SECONDS),
                     f2.get(1, TimeUnit.SECONDS));
```

Loom: Virtual Threads

A classical business use case:

```
if (!userService.exists(name)) {
    User user = new User(name);
    userService.create(user);
}
User user = userService.findByName(name);
var cart = cartService.loadCartFor(user);
var totalPrice =
        cart.items().stream()
                .mapToInt(Item::price)
                .sum();
var transactionId = paymentService.pay(user, totalPrice);
boolean sent = emailService.send(user, cart, transactionId);
```

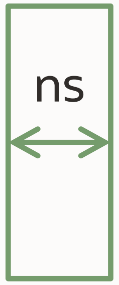
Loom: Virtual Threads

```
CompletableFuture.supplyAsync(
        () -> userService.exists(name))
        .thenCompose(
                userExists -> {
                    if (!userExists) {
                        User user = new User(name);
                        return supplyAsync(() -> userService.create(user));
                    } else {
                        return CompletableFuture.completedFuture(true);
                    }
                }
        )
        .thenCompose(
                userCreated -> {
                    if (userCreated) {
                        return supplyAsync(() -> userService.findByName(name));
                    } else {
                        return CompletableFuture.completedFuture(null);
                    }
                }
        )
        .thenCompose(
                user -> {
                    if (user != null) {
                        return CompletableFuture.supplyAsync(() -> cartService.loadCartFor(user))
                                .thenCompose(cart -> {
                                    int totalPrice = cart.items().stream().mapToInt(Item::price).sum();
                                    return CompletableFuture.supplyAsync(() -> paymentService.pay(user, totalPrice))
                                            .thenCompose(transactionId -> CompletableFuture.supplyAsync(() -> emailService.send(user, cart, transactionId)));
                                });
                    } else {
                        return CompletableFuture.completedFuture(null);
                    }
                }
        );
```

Concurrency Issues

Why is it bad to block?

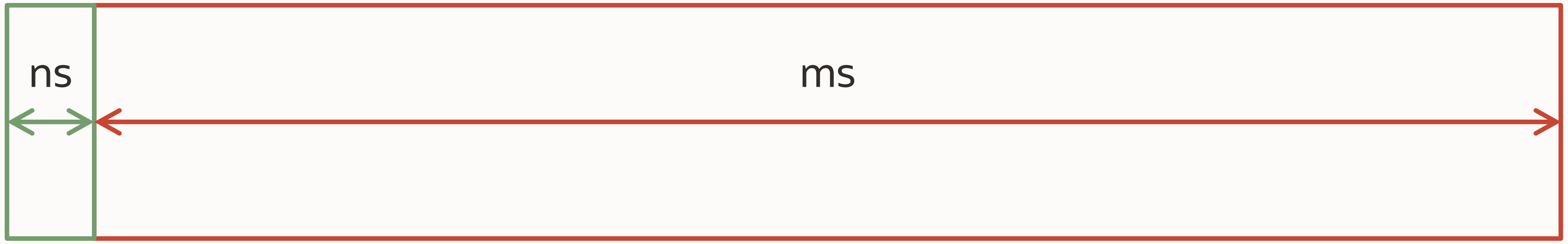
```
Json request            = buildContractRequest(id);
String contractJson = contractServer.getContract(request);
Contract contract     = Json.unmarshal(contractJson);
```



Concurrency Issues

Why is it bad to block?

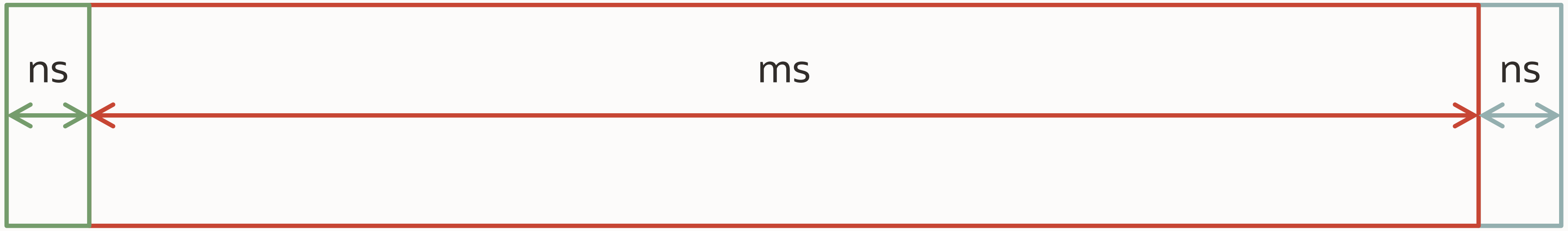
```
Json request          = buildContractRequest(id);
String contractJson = contractServer.getContract(request);
Contract contract     = Json.unmarshal(contractJson);
```



Concurrency Issues

Why is it bad to block?

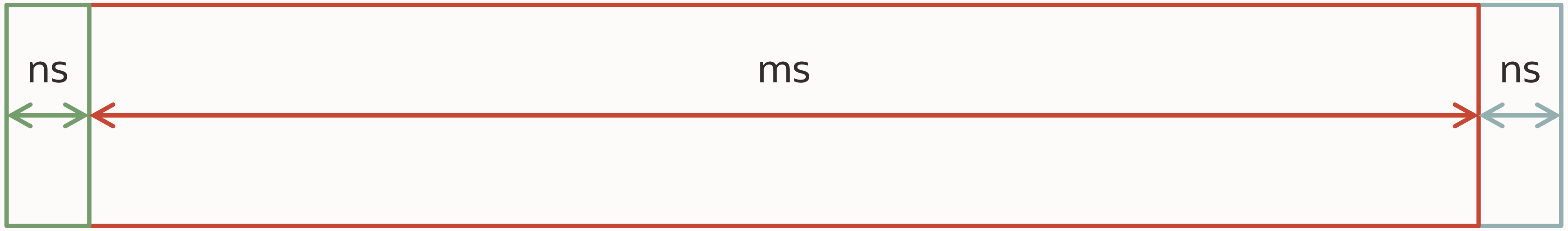
```
Json request          = buildContractRequest(id);
String contractJson = contractServer.getContract(request);
Contract contract     = Json.unmarshal(contractJson);
```



Concurrency Issues

Why do we need to write asynchronous code based on callbacks?

Because a Thread is idle 99.9999% of the time when you are processing I/O data:



How many threads do you need to have 100% CPU ?

Concurrency Issues

A `java.lang.Thread` is a wrapper on a kernel (or platform) thread

It needs:

- ~1ms to start
- ~2MB of memory to store its stack
- context switching costs ~0,1ms

You can only have several thousands of them

Concurrency Issues

It means that, instead of beeing busy 0.0001% of the time
Your CPU is now busy about 1% of the time

It’s not enough!

Concurrency Issues

At this point, you have two solutions:

1) You give 1k – 1M tasks to each thread. This is what asynchronous frameworks are doing.
   And it comes with a high maintenance cost!

Concurrency Issues

```java
CompletableFuture.supplyAsync(
        () -> userService.exists(name))
        .thenCompose(
                userExists -> {
                    if (!userExists) {
                        User user = new User(name);
                        return supplyAsync(() -> userService.create(user));
                    } else {
                        return CompletableFuture.completedFuture(true);
                    }
                }
        )
        .thenCompose(
                userCreated -> {
                    if (userCreated) {
                        return supplyAsync(() -> userService.findByName(name));
                    } else {
                        return CompletableFuture.completedFuture(null);
                    }
                }
        )
        .thenCompose(
                user -> {
                    if (user != null) {
                        return CompletableFuture.supplyAsync(() -> cartService.loadCartFor(user))
                                .thenCompose(cart -> {
                                    int totalPrice = cart.items().stream().mapToInt(Item::price).sum();
                                    return CompletableFuture.supplyAsync(() -> paymentService.pay(user, totalPrice))
                                            .thenCompose(transactionId -> CompletableFuture.supplyAsync(() -> emailService.send(user, cart, transactionId)));
                                });
                    } else {
                        return CompletableFuture.completedFuture(null);
                    }
                }
        );
```

Is reactive programming a good solution?

Concurrency Issues

At this point, you have two solutions:

1) You give 1k – 1M tasks to each thread. This is what asynchronous frameworks are doing.
   And it comes with a high maintenance cost!

2) Or you create another model of thread, that is 1000 lighter, so that you can have 1M of them

Loom: Virtual Threads

A Platform thread:
- takes ~1ms to start
- consumes ~20MB
- context switching ~0,1ms

A Virtual thread:
- takes ~1μs to start
- consumes ~200kB

A virtual thread is a thread
- Race conditions, visibility, locking, … are the same
- But lighter by a factor of 1000

Loom: Virtual Threads

A virtual thread is a wrapper on your task, that can be mounted and unmounted from a platform thread

Everytime a virtual thread blocks (I/O, synchronization, …), it is unmounted from its platform thread

Blocking a virtual thread is fine, because it does not block any platform thread

Loom: Virtual Threads

Launching a virtual thread:

```
Runnable task =
   () ->
      System.out.println("I am running in " +
                             Thread.currentThread().getName());

Thread thread = Thread.ofVirtual()
                         .unstarted(task);

thread.start();
thread.join();
```

Loom: Virtual Threads

Creating a pool of virtual threads:

```
ExecutorService service =
   Executors.newVirtualThreadPerTaskExecutor();

var future = service.submit(task);
```

Loom: Virtual Threads

Running a task in a Virtual Thread
Is more expensive than running it in a Platform Thread

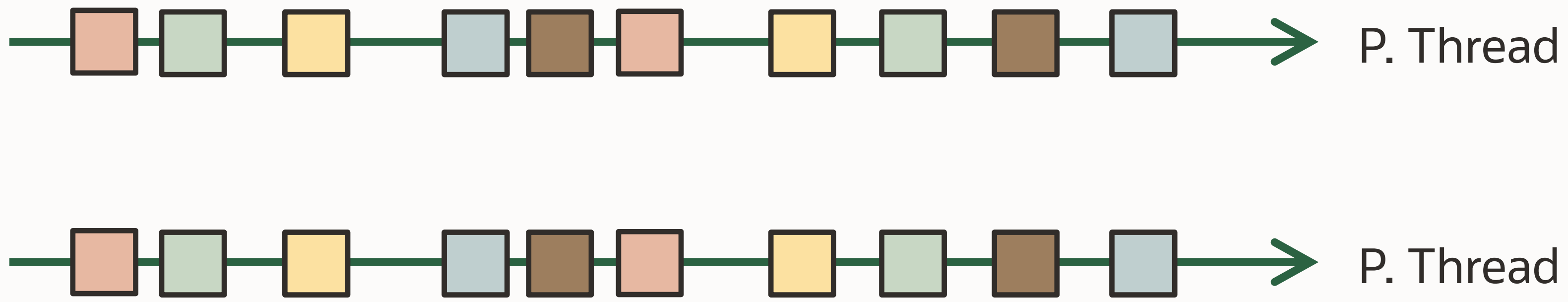
Running non-blocking, in-memory task is useless!

Virtual Threads are meant to run blocking (I/O) tasks

From the Platform Thread Point of View

With Virtual Threads

With React. Progr.



From the Platform Thread Point of View

1) The performances of Virtual Threads and Reactive Programming should be the same
   The differences come from the frameworks
2) In both cases, blocking a Platform Thread is a major performance hit
3) With Reactive Programming, not blocking a Platform Thread is the responsibility of the code!
4) With Virtual Threads, it is handled by the API (Java I/O, NIO)

Wrapping up Virtual Threads

1) Are cheap to create, you can have a million of them
   Their memory consumption is low
   And will improve over time
2) Are used to run blocking code
   If you don’t plan to block them, don’t use them!
3) Prevent platform threads to be blocked
   No need to write asynchronous / reactive code anymore

Loom: Structured Concurrency (prev 21)

StructuredConcurrency brings new patterns of code to leverage virtual threads and avoid asynchronous code

Loom: Structured Concurrency (prev 21)

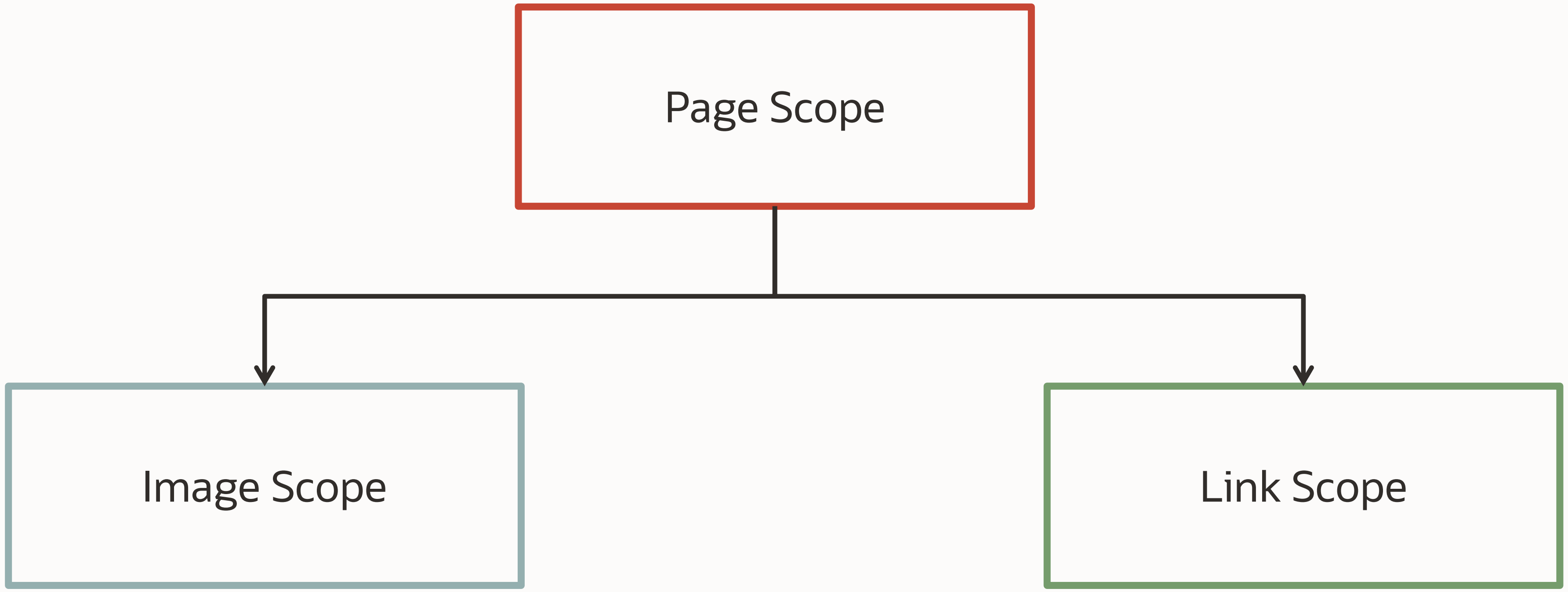
```java
try (var scope = new StructuredTaskScope()) {

   var sup1 = scope.fork(() -> readImages());
   var sup2 = scope.fork(() -> readText());
   var sup3 = scope.fork(() -> readLinks());

   scope.join();

   // do something with sup1, sup2, ...
   return result;
}
```

Loom: Structured Concurrency (prev 21)



Loom: Structured Concurrency (prev 21)

Exiting the try-with-resource block cleans up everything

No more loose thread!

ScopedValue (prev 21)

An alternative model for ThreadLocal variables

ThreadLocal are supported by virtual threads

But you can do better!

ScopedValue (prev 21)

What is wrong with ThreadLocal?

They are mutable
They can be inherited

They are bound to a thread, and a thread is not bound
ScopedValues want to be bounded!

ScopedValue (prev 21)

ScopedValues are non-modifiable

They are not bound to a particular thread

```
ScopedValue<String> key = new ScopedValue.newInstance();

ScopedValue.where(key, "KEY_1")
           .run(() -> doSomethingSmart());

ScopedValue.where(key, "KEY_2")
           .run(() -> doSomethingSmart())
           .run(() -> soSomethingSmarter());
```

ScopedValue (prev 21)

ScopedValues are non-modifiable
They are not bound to a particular thread

```
void doSomethingSmart() {
   if (key.isBound()) {
      String value = key.get();
      ...
   } else {
      throw new IllegalStateException("Key is not bound");
   }
}
```

ScopedValue (prev 21)

ScopedValues are NOT transmitted to threads or virtual threads

Because a ScopedValue wants to be bound

A StructuredTaskScope IS bound

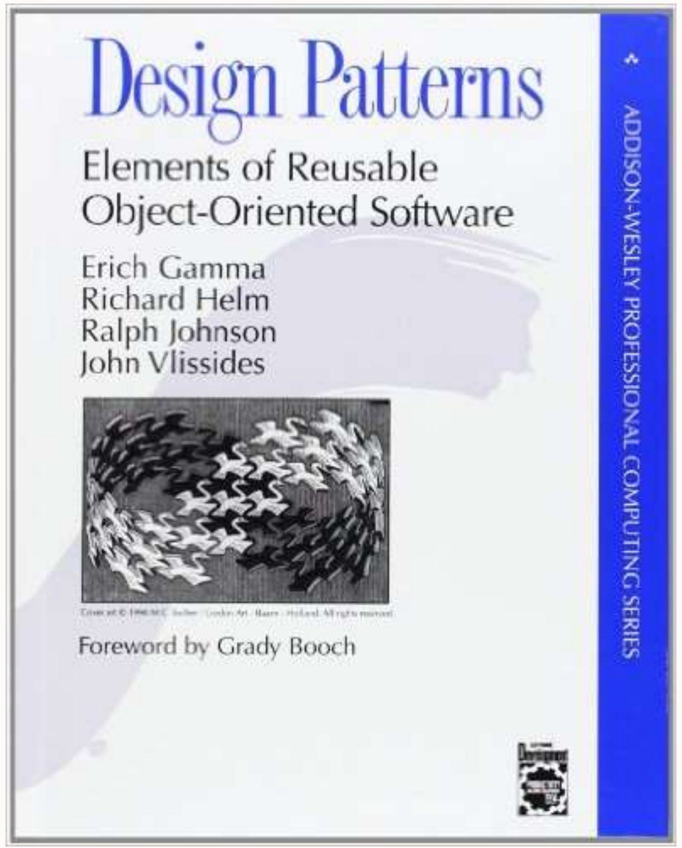
ScopedValues are transmitted to StructuredTaskScope

Loom: Current Plan for the JDK 21

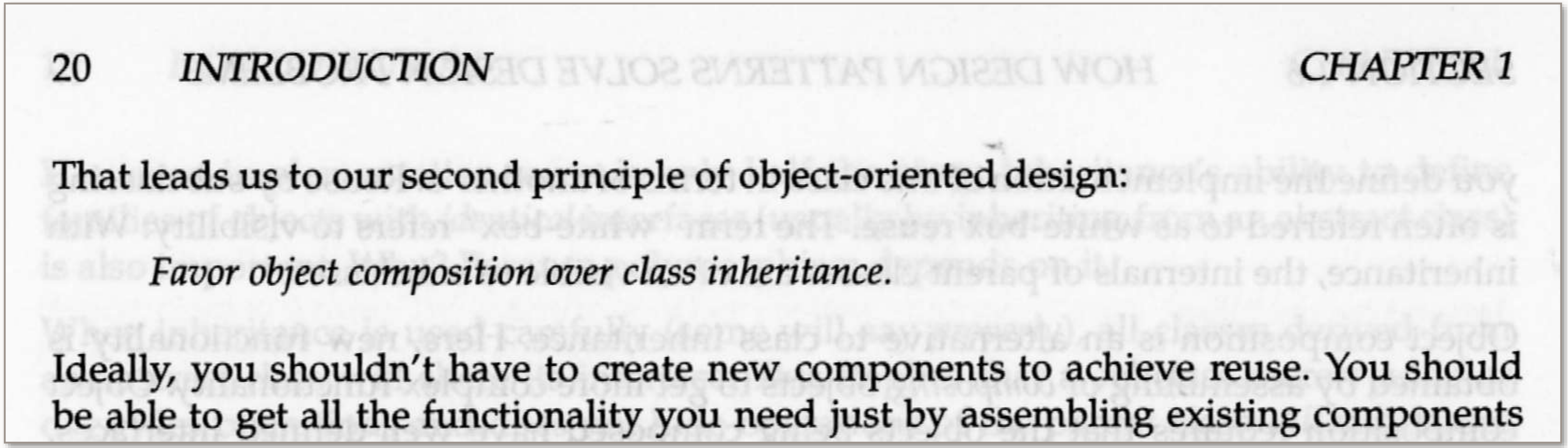
	JDK 21	JDK 22
Virtual Threads	Final	
Structured Concurrency	Preview	?
Scoped Values	Preview	?

Valhalla

Valhalla


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Valhalla



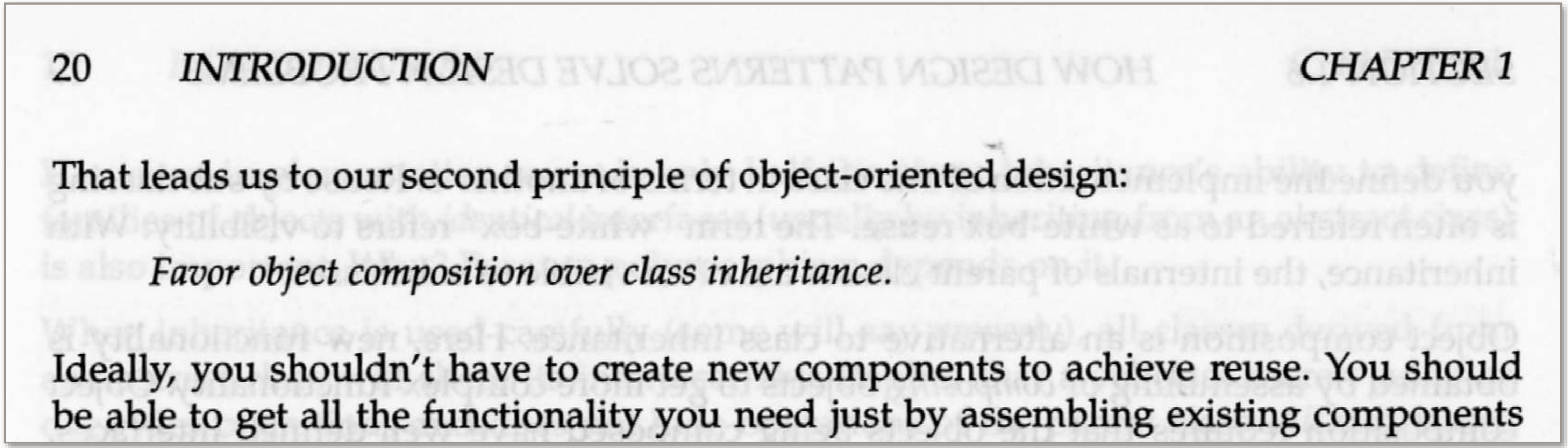
20 INTRODUCTION CHAPTER 1

That leads us to our second principle of object-oriented design:

Favor object composition over class inheritance.

Ideally, you shouldn't have to create new components to achieve reuse. You should be able to get all the functionality you need just by assembling existing components

Valhalla



20 INTRODUCTION CHAPTER 1

That leads us to our second principle of object-oriented design:

Favor object composition over class inheritance.

Ideally, you shouldn't have to create new components to achieve reuse. You should be able to get all the functionality you need just by assembling existing components

Why does nobody follow this principle?

Valhalla: Value Classes

Summing populations with ints

```
int[] populations = {...};
```

```
int totalPopulation = 0;
for (int population: populations) {
   totalPopulation += population;
}
```

```
int totalPopulation = Arrays.stream(populations).sum();
```

Valhalla: Value Classes

Summing populations with records

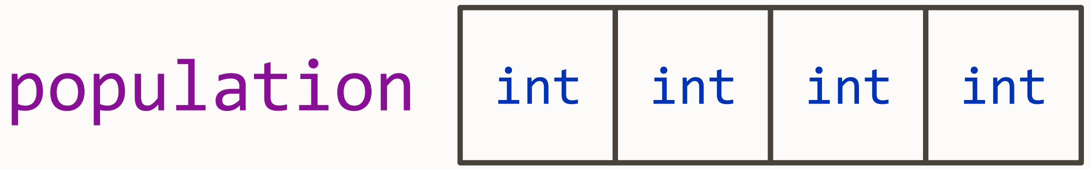
```
record Population(int population) {}
Population[] populations = {...};
```

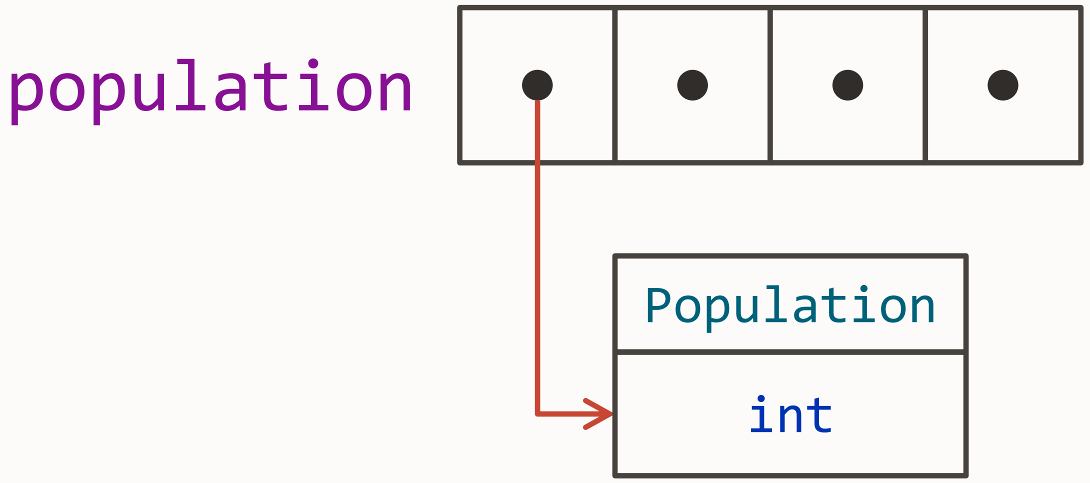
```
Population totalPopulation = Population.zero();
for (Population population: populations) {
   totalPopulation = totalPopulation.add(population);
}
```

```
Population totalPopulation =
   Arrays.stream(populations)
         .reduce(Population.zero(), Population::add);
```

Layout in Memory

Pointer chasing is a performance hit





int[]

Population[]

Valhalla: Value Classes

Creating histograms

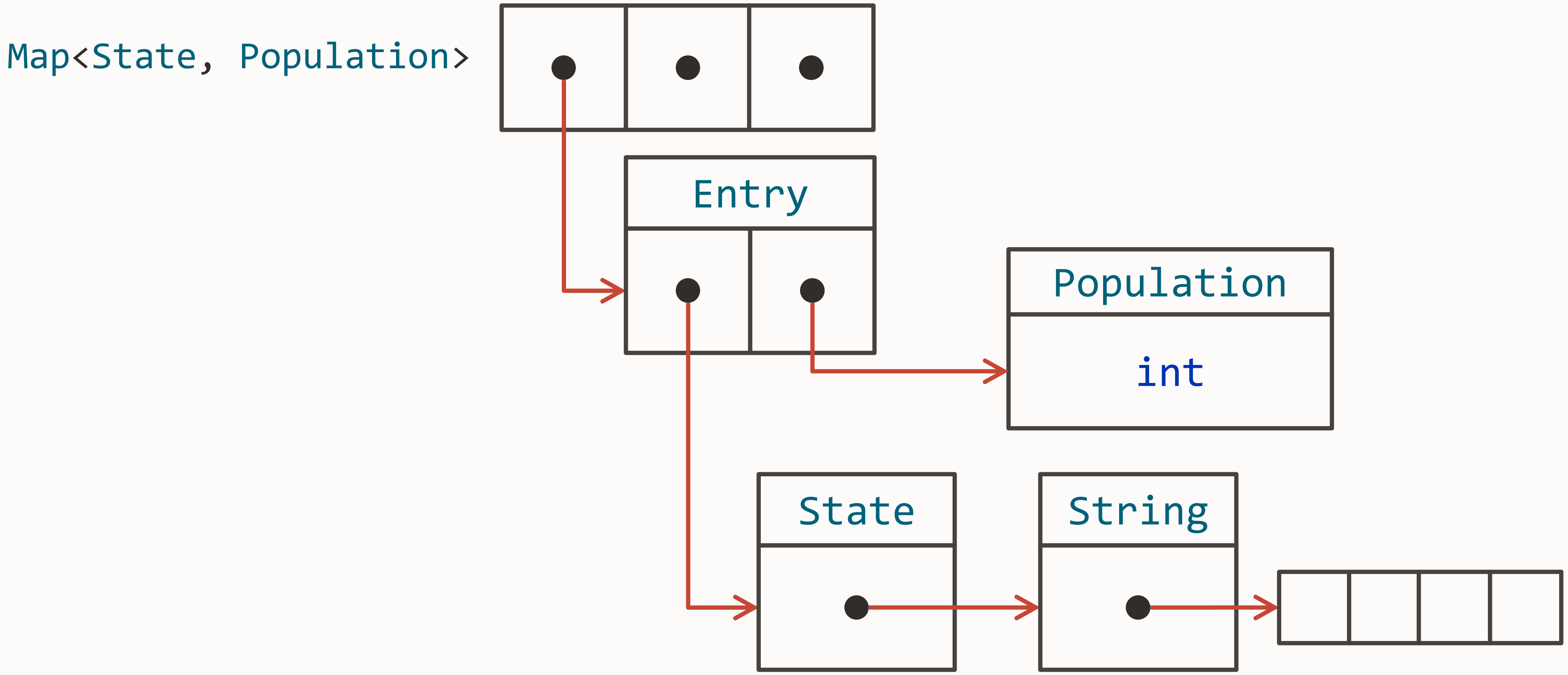
```
// this is the histogram of population by state
Map<String, Integer> populationByState = ...;
```

```
record State(String name, List<City> cities) {}
record City(String name, Population population) {}
record Population(int population) {}

Map<State, Population> populationByState = ...;
```

Layout in Memory

Pointer chasing is a performance hit



Valhalla

1st goal: make it so that you do not have to choose between readable code and performances

Make abstraction (almost) free

Codes like a class, Works like an int

Valhalla: Value Classes

Value class: you need to give up on something!

- is implicitely final
- has instance field that are final
- does not have an address (no synchronization)
- == compares the fields

Records can be declared `value record`

Valhalla: Value Classes

```
value record Population (int population) {}

value record City(String name, Population population) {}
```

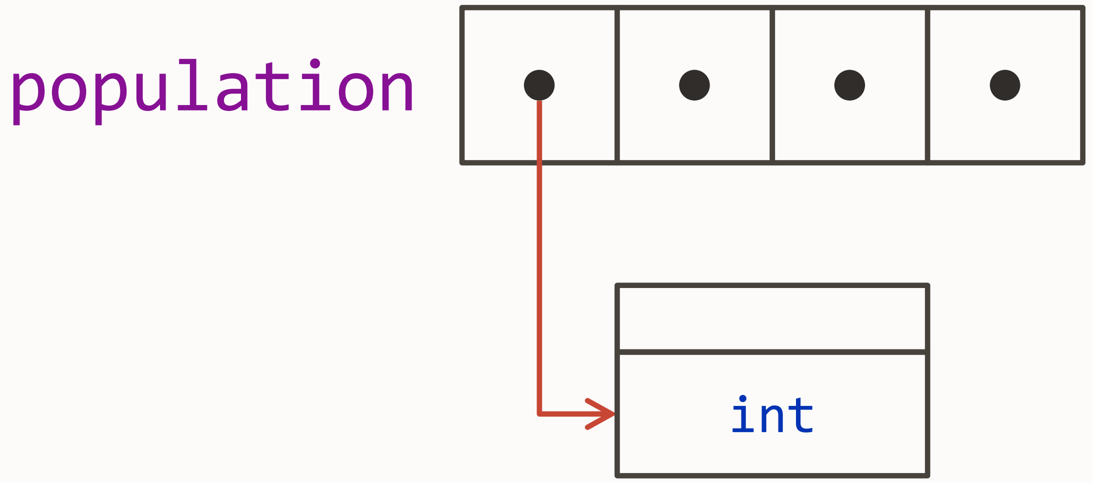
Valhalla: Value Classes

Because they do not need to be stored as an object with an address, value objects offer better performances:

- they can be stored in contiguous zones of the memory
- they can be inlined in variables or in registers
- no pointer chasing to access them

Layout in Memory

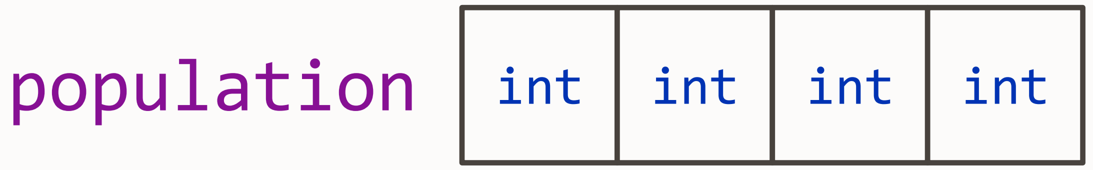
Extra pointers / space in memory



Population[] object version

Layout in Memory

No more pointer, no more header



Population[] value type version

Valhalla: Primitive Type

What is a primitive type?

- It is not an object, it does not have an address
- == compares the value
- It does not make sense to modify a *value*
- It cannot be null

So it needs a *default value* (that is not null)

Valhalla: Primitive Type

```
record Population(int population) {} // suppose this is a primitive type
```

A primitive type cannot be null
So the following declares an array of `new Population(0)`

```
Population[] pops = new Population[10]; // filled with default values
```

Valhalla: Primitive Type

```
Population[] pops = new Population[10]; // filled with default values
```

But you can wrap an array with a list:

```
List<Population> pops = Arrays.asList(new Population[10]);

pops.set(0, null); // this is legal code
```

You need a way to declare that something cannot be null

Valhalla: Primitive Type

Defining custom primitive types requires:

- to be able to define default values
- to modify the type system, so that you can declare that a reference cannot be null, where you are using it

Valhalla: Primitive Type

Declaring a default instance value record
= a value record that can be declared not to be null

```
public value record Population(int population) {

   public default Population();

   public Population {
      if (x < 0) throw new IllegalArgumentException("Nope!");
   }
}
```

Valhalla: Primitive Type

When using this type, you can declare that it cannot be null. The default instance will be used instead.

```
Population[] pops = new Population[10]; // filled with null values
```

```
Population![] pops = new Population![10]; // filled with default values
```

```
value record City(String name, Population! population) {}
```

Valhalla: Primitive Type

When using this type, you can declare that it cannot be null. The default instance will be used instead.

```
List<Population!> populations = Arrays.asList(new Population![10]);

populations.set(0, null); // this is not legal anymore
```

Valhalla: Primitive Type

With that in mind, all these have the same in-memory representation:

- `List<Integer!>` and `List<int>`
- `314` and `Integer.valueOf(314)`

So this kind of syntax becomes possible:

```
double d = 314.doubleValue();
Supplier<String> function = 314::toString;
```

Valhalla: Primitive Type

Modeled by default instance value classes

As such, they have no identity

It is a bare sequence of field values, without any headers of extra pointers

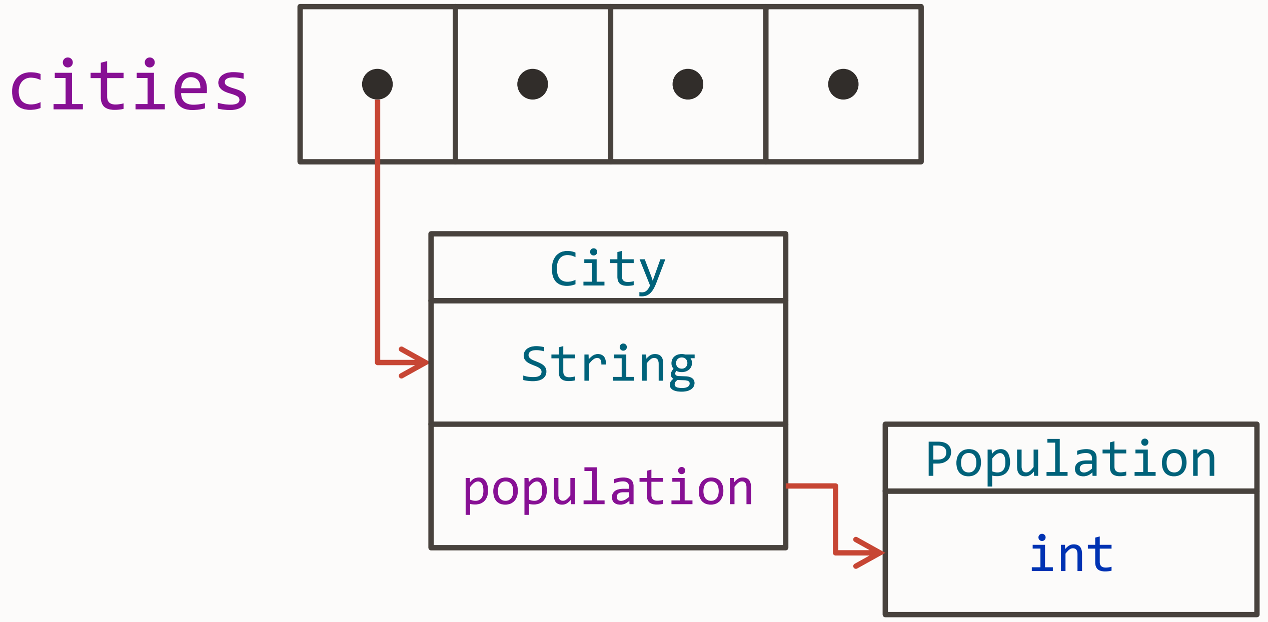
== compares the field values

They have a default value

You can decide to have null values or not

Layout in Memory

Extra pointers / space in memory



Object version

Layout in Memory

Can inline everything

`cities`

`String`	`String`	`String`	`String`
`int`	`int`	`int`	`int`

City![] / Population! version

Valhalla: Primitive Type

Identity class
- nullable
- non-tearable
- boxing not needed
- non flattened

Value class
- nullable
- non-tearable
- boxing not needed
- flattened
    - on the stack
    - on the heap?

Default instance
- default value
- tearable?
- can be boxed
- flattened
    - on the stack
    - on the heap

Valhalla: Primitive Type

The well-known wrapper classes are to be converted to primitive classes

More value classes:

`Optional`, `LocalDate`, …

Valhalla: Current Plan for the JDK 2?

	JDK 21	JDK 22
Value types	EA builds	?
Primitive Types	EA builds	?
And the rest…	EA builds	?

Have fun with Amber, Loom, Valhalla, and the others!

Amber, Loom, Valhalla

Links

Links and References

Amber

Switch Expression: http://openjdk.java.net/jeps/361

Record: http://openjdk.java.net/jeps/395

Sealed Classes: http://openjdk.java.net/jeps/409

Pattern Matching for instanceof: http://openjdk.java.net/jeps/394

Pattern Matching for Switch (3rd preview): http://openjdk.java.net/jeps/427

Record Patterns: http://openjdk.java.net/jeps/405

Links and References

Loom

Virtual Threads (preview): http://openjdk.java.net/jeps/425

Structured Concurrency (incubator): http://openjdk.java.net/jeps/428

Links and References

Valhalla

Universal Generics: http://openjdk.java.net/jeps/8261529

Value Objects: https://openjdk.org/jeps/8277163

Primitive Classes (preview): https://openjdk.java.net/jeps/401

Classes for Basic Primitives (preview): https://openjdk.java.net/jeps/402

Links and References

Panama

Vector API (4th incubator) http://openjdk.java.net/jeps/426

Foreign Function & Memory API (preview): http://openjdk.java.net/jeps/424