ORACLE

How Amber is Driving Java's Evolution

Billy Korando

Oracle - Java Developer Advocate ☕🥑
@BillyKorando

Important Notes

- Ask questions
- Reach out:
  Twitter: @BillyKorando
  Email: billy.korando@oracle.com
- Link to slides 👉 https://github.com/wkorando/how-amber-is-driving-javas-evolution

- https://dev.java
- https://inside.java
- Inside Java Podcast
- #SipOfJava
- https://youtube.com/java
- Inside Java Newscast
- #JEPCafé

Project Amber

http://openjdk.java.net/projects/amber/

“The goal of Project Amber is to explore and incubate smaller, productivity-oriented Java language features…”

Other Active OpenJDK Projects

Project Valhalla
Project Panama
Project Loom

Other Active OpenJDK Projects

Project Valhalla

Improve the memory model map
Introduction of Value Types

Value Types often defined with “val” in langauges

Thus “Project **Val**halla”

Other Active OpenJDK Projects

Project Panama

Improve the interface between the JVM and “native” interfaces

The Panama Canal famously connected the Atlanta and Pacific Oceans

Thus “Project Panama” connecting “two oceans”

Other Active OpenJDK Projects

Project Loom

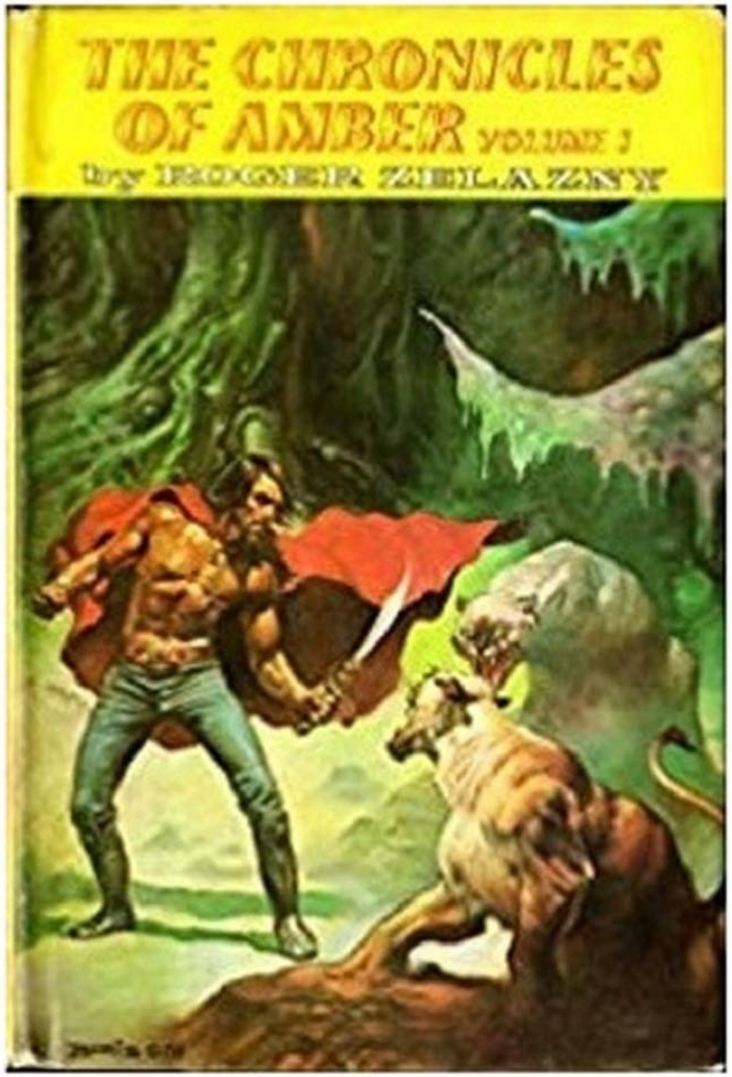
Improve the handling of threads in Java (introduction of virtual threads)

Looms are used to weave threads together

Thus “Project Loom”

Project Amber

Project Amber



Ritual of Passage was:

“Walking the Pattern”

Project Amber

http://openjdk.java.net/projects/amber/

“The goal of Project Amber is to explore and incubate smaller, productivity-oriented Java language features…”

Other changes delivered under Amber:

Local Variable Type Inference (var)

Text Blocks

Agenda

- Key Language Changes in 11-17
- Themes of Language Changes
- The Future of Java

Let's Review:
Key language changes in 11-17

Key Language Changes

- Updates to `switch`
- Pattern matching for `instanceof`
- Records
- Sealed classes

Switch Updates

Added in Java 14
JEP 361

Switch Updates

```
switch(d){
    case 1:
     System.out.println("Sunday");
     break;
    case 2:
     System.out.println("Monday");
     break;
    case 3:
     System.out.println("Tuesday");
     break;
    case 4:
     System.out.println("Wednesday");
     break;
    case 5:
     System.out.println("Thursday");
     break;
    case 6:
     System.out.println("Friday");

    case 7:
     System.out.println("Saturday");
     break;
}
```


Switch Updates

```
switch(d){
    case 1 -> System.out.println("Sunday");
    case 2 -> System.out.println("Monday");
    case 3 -> System.out.println("Tuesday");
    case 4 -> System.out.println("Wednesday");
    case 5 -> System.out.println("Thursday");
    case 6 -> System.out.println("Friday");
    case 7 -> System.out.println("Saturday");
}
```

Switch Updates

```
String day = switch(d){
     case 1 -> "Sunday";
     case 2 -> "Monday";
     case 3 -> "Tuesday";
     case 4 -> "Wednesday";
     case 5 -> "Thursday";
     case 6 -> "Friday";
     case 7 -> "Saturday";
     default -> throw new IllegalArgumentException();
}
```

Switch Updates

```
String day = switch(d){
     case 1 -> "Sunday";
     case 2 -> "Monday";
     case 3 -> "Tuesday";
     case 4 -> "Wednesday";
     case 5 -> "Thursday";
     case 6 -> {
       System.out.println("Ladies and Gentlemen, the Weekend");
       yield "Friday";
     }
     case 7 -> "Saturday";
     default -> throw new IllegalArgumentException();
}
```


Switch Updates

```
String day = switch(d){
    case 1:
        yield "Sunday";
    case 2:
        yield "Monday";
    case 3:
        yield "Tuesday";
    case 4:
        yield "Wednesday";
    case 5:
        yield "Thursday";
    case 6:
        yield "Friday";
    case 7:
        yield "Saturday";
    default:
        throw new IllegalArgumentException();
}
```


Switch Updates

```
enum DaysOfWeek {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;
}

DaysOfWeek dayOfWeek = [someDayOfWeek]

String day = switch(dayOfWeek){
     case SUNDAY -> "Sunday";
     case MONDAY -> "Monday";
     case TUESDAY -> "Tuesday";
     case WEDNESDAY -> "Wednesday";
     case THURSDAY -> "Thursday";
     case FRIDAY -> "Friday";
     case SATURDAY -> "Saturday";
}
```


Switch Updates

```
enum DaysOfWeek {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;
}

String day = switch(dayOfWeek){
    case SUNDAY :
        yield "Sunday";
    case MONDAY :
        yield "Monday";
    case TUESDAY :
        yield  "Tuesday";
    case WEDNESDAY:
        yield "Wednesday";
    case THURSDAY:
        yield "Thursday";
    case FRIDAY:
        yield "Friday";
    case SATURDAY:
        yield "Saturday";
};
```


Pattern Matching for Instanceof

Added in Java 16
JEP 394

Pattern Matching for Instanceof

```
Object actuallyAString = "I'm actually a string!";

if(actuallyAString instanceof String) {//Test if actuallyAString is a String
    String nowImAString = //Assign actuallyAString to a variable
    (String) actuallyAString; //Convert actuallyAString to a String

    System.out.println(nowImAString);
}
```


Pattern Matching for Instanceof

```
Object actuallyAString = "I'm actually a string!";

if(actuallyAString instanceof String nowImAString) {
    System.out.println(nowImAString);
}
```

Predicate

Pattern Variable

Pattern Matching for Instanceof

```
Object actuallyAString = "I'm actually a string!";

if(actuallyAString instanceof String nowImAString) {
    System.out.println(nowImAString);
}

System.out.println(nowImAString); //Compiler error, nowImAString not in scope

boolean isAString = (actuallyAString instanceof String nowImAString);

System.out.println(nowImAString); //Compiler error, nowImAString not in scope
```

In scope where compiler knows pattern variable is set

Pattern Matching for Instanceof

```
Object actuallyAString = "I'm actually a string!";

if (!(actuallyAString instanceof String nowImAString)) {
//...
} else {
    System.out.println(nowImAString);
}

if (!(actuallyAString instanceof String nowImAString)) {
    throw new IllegalArgumentException("Must be a string!");
}

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

Records

Added in Java 16
JEP 395

Records

```
String firstName1 = "Billy";
String lastName1 = "Korando";
String title1 = "Java Developer Advocate";
String twitterHandle1 = "@BillyKorando";

String firstName2 = "Sharat";
String lastName2 = "Chander";
String title2 = "Java Developer Advocate";
String twitterHandle2 = "@Sharat_Chander";

class Person{
    private String firstName;
    private String lastName;
    private String title;
    private String twitterHandle;
    public Person(String firstName, String lastName, String title, String twitterHandle) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.title = title;
        this.twitterHandle = twitterHandle;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((firstName == null) ? 0 : firstName.hashCode());
        result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
        result = prime * result + ((title == null) ? 0 : title.hashCode());
        result = prime * result + ((twitterHandle == null) ? 0 : twitterHandle.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (firstName == null) {
            if (other.firstName != null)
                return false;
        } else if (!firstName.equals(other.firstName))
            return false;
        if (lastName == null) {
            if (other.lastName != null)
                return false;
        } else if (!lastName.equals(other.lastName))
            return false;
        if (title == null) {
            if (other.title != null)
                return false;
        } else if (!title.equals(other.title))
            return false;
        if (twitterHandle == null) {
            if (other.twitterHandle != null)
                return false;
        } else if (!twitterHandle.equals(other.twitterHandle))
            return false;
        return true;
    }
    @Override
    public String toString() {
        return "Person [firstName=" + firstName + ", lastName=" + lastName + ", title=" + title
                + ", twitterHandle=" + twitterHandle + "]";
    }
}

var persons = Stream.of(new Person(firstName1, lastName1, title1, twitterHandle1),
new Person(firstName2, lastName2, title2, twitterHandle2));

persons.forEach(System.out::println);
```

Records

```
String firstName1 = "Billy";
String lastName1 = "Korando";
String title1 = "Java Developer Advocate";
String twitterHandle1 = "@BillyKorando";

String firstName2 = "Sharat";
String lastName2 = "Chander";
String title2 = "Java Developer Advocate";
String twitterHandle2 = "@Sharat_Chander";

record Person(String firstName, String lastName, String title, String twitterHandle) {}

var persons = Stream.of(new Person(firstName1, lastName1, title1, twitterHandle1),
        new Person(firstName2, lastName2, title2, twitterHandle2));

persons.forEach(System.out::println);
```

Records

- Transparent modeling of data as data

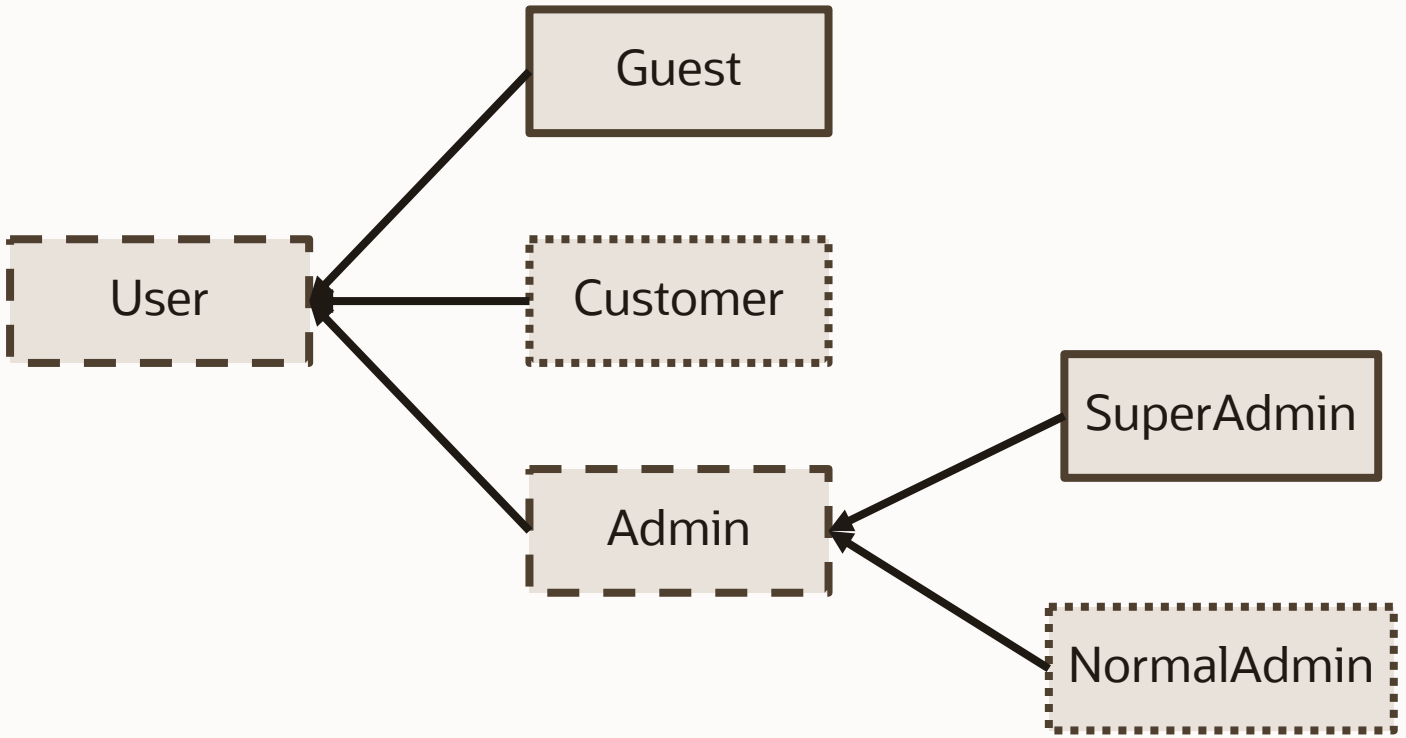
```
record Person(String firstName, String lastName, String title, String twitterHandle) {}
```

- Superclass always `java.lang.Record`
- Cannot be extended, abstract, and implicitly final
- All fields are final (shallowly immutable)
- Cannot declare instance fields, field initializers, instance initializers
- Accessors, `hashCode()`, `toString()`, `equals()`, automatically generated, but can be overwritten
- Fundamentally just a class

Sealed Classes

Added in Java 17
JEP 409

Sealed Classes



Sealed Classes

```
public abstract sealed class User
permits Admin, Customer, Guest{

}

public sealed class Admin
extends User permits SuperAdmin, NormalAdmin{

}

public final class Guest extends User {

}

public non-sealed class Customer extends User {

}
```


Sealed Classes & Records

```
public sealed interface User{}

public record Admin(...) implements User{}
public record Guest(...) implements User{}
public record Customer(...) implements User{}
```


Project Amber: Timeline

	Java 10	Java 11	Java 12	Java 13	Java 14	Java 15	Java 16	Java 17
Local Variable Type Inference - `var`	Standard	→	→	→	→	→	→	→
Local Variable Syntax for Lambda Parameters		Standard	→	→	→	→	→	→
Switch Expressions			Preview	2nd Preview	Standard	→	→	→
Text Blocks				Preview	2nd Preview	Standard	→	→
Records					Preview	2nd Preview	Standard	→
Pattern Matching for `instanceof`					Preview	2nd Preview	Standard	→
Sealed Classes						Preview	2nd Preview	Standard

Themes

1. More expressive
   - Meaning can be derived more easily from code
2. Safer
   - Compiler warnings
   - Address some security issues (serialization)
   - Fewer concerns with nulls
3. Reducing verbosity/ceremony
   - Not *just* about typing, but reducing opportunities for bugs

“Making it easier to program in plain data”

Brian Goetz – Java Language Architect
https://www.youtube.com/watch?v=krmW1wcFvcE

The Future of Java

Near Future

Pattern Matching for Switch

Currently in Preview 2
JEP 420
Soon in Preview 3
JEP 8282272 (Draft)

Pattern Matching for Switch

```
return switch (o) {
      case Integer i -> String.format("int %d", i);
      case Long l    -> String.format("long %d", l);
      case Double d  -> String.format("double %f", d);
      case String s  -> String.format("String %s", s);
      default        -> o.toString();
};
```

Pattern Matching for Switch

```
return switch (o) {
     case Integer i when i > 0 && i / 2 -> System.out.println("Positive & even" + i);
     case Integer i when i > 0 -> System.out.println("Positive & odd" + i);
     case Integer i -> System.out.println("Zero or negative" + i);
     default        -> o.toString();
};
```


Pattern Matching for Switch

```
switch (s) {
    case null          -> System.out.println("Oops");
    case "Foo", "Bar" -> System.out.println("Great");
    default            -> System.out.println("Ok");
}
```


Pattern Matching for Switch

```
public sealed interface User{}

public record Admin(...) implements User{}
public record Guest(...) implements User{}
public record Customer(...) implements User{}

User u = ...

User someUser = switch(u){
    case Admin a -> ...;
    case Guest g -> ...;
    case Customer c -> ...;
}
```


Pattern Matching for Switch

```
public sealed interface User{}

public record Admin(...) implements User{}
public record Guest(...) implements User{}
public record Customer(...) implements User{}

User u = ...

switch(u){  //Error not exhaustive!
    case Admin a -> ...;
    case Customer c -> ...;
}
```


Record Patterns

Currently in candidate status
JEP 405

Record Patterns

```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}

static void printColorOfUpperLeftPoint(Rectangle rectangle) {
    if(rectangle != null){
        ColoredPoint ul = rectangle.ul();
        if (ul != null) {
            Color c = ul.c();
            System.out.println(c);
        }
    }
}
```


Record Patterns

```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}

static void printColorOfUpperLeftPoint(Rectangle rectangle) {
    if(rectangle instaceof Rectangle r){
        ColoredPoint ul = r.ul();
        if (ul != null) {
            Color c = ul.c();
            System.out.println(c);
        }
    }
}
```


Record Patterns

```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}

static void printColorOfUpperLeftPoint(Rectangle rectangle) {
    if(rectangle instaceof Rectangle(ColoredPoint ul, ColoredPoint lr){
        Color c = ul.c();
        System.out.println(c);
    }
}
```


Record Patterns

```
record Point(int x, int y) {}
enum Color { RED, GREEN, BLUE }
record ColoredPoint(Point p, Color c) {}
record Rectangle(ColoredPoint upperLeft, ColoredPoint lowerRight) {}

static void printColorOfUpperLeftPoint(Rectangle rectangle) {
    if(rectangle instaceof Rectangle(ColoredPoint(Point p, Color c), ColoredPoint lr){
        System.out.println(c);
    }
}
```


A Bit Further Out Future

Array Patterns

```
String[] someStrings = new String[]{
"foo",
"bar"};

if(someStrings instanceof String[]{String s1, String s2}){
    System.out.println(s1);
    System.out.println(s2);
}
```


Array Patterns

```
String[][] multiDimensionalArray ...

if(multiDimensionalArray instanceof
    String[][]{{var s1, var s2},{var s3, var s4}}){
      ...
}
```


Array Patterns

```
public sealed interface User{}

public record Admin(String access) implements User{}
public record Guest(String id) implements User{}
public record Customer(int loyaltyNumber) implements User{}

User u = ...

User someUser = switch(u){
    case Admin(var a) -> ...;
    case Guest(var id) -> ...;
    case Customer(var ln) -> ...;
}
```

Enhanced Arrays

```
int[] someNums = {...};
int n = someNums.length;

if(someNums instanceof int[]{[n-2] -> int x, [n-1] -> int y}){
    System.out.println("Sum of last two elements: " + (x+y));
}
```


Don’t Care Patterns

```
void int getXfromPoint(Object o) {
    if (o instanceof Point(var x, _)){
        return x;
    }
    return -1;
}
```

Don’t Care Patterns

```
void int getXfromPoint(Object o) {
    if (o instanceof Point(var x)){
        return x;
    }
    return -1;
}
```


Pattern Matching in foreach Loops

```
for (Entry e : map.entrySet()){
        System.out.printf("%s: %s%n", e.key(), e.value());
```


Pattern Matching in foreach Loops

```
for (Map.Entry(var key, var value) : map.entrySet())
        System.out.printf("%s: %s%n", key, value);
```


Early Access Builds

https://jdk.java.net/19/
https://jdk.java.net/loom/
https://jdk.java.net/panama/
https://jdk.java.net/valhalla/

Thank you

Twitter: @BillyKorando
Email: billy.korando@oracle.com

Slides: https://github.com/wkorando/how-amber-is-driving-javas-evolution


ORACLE