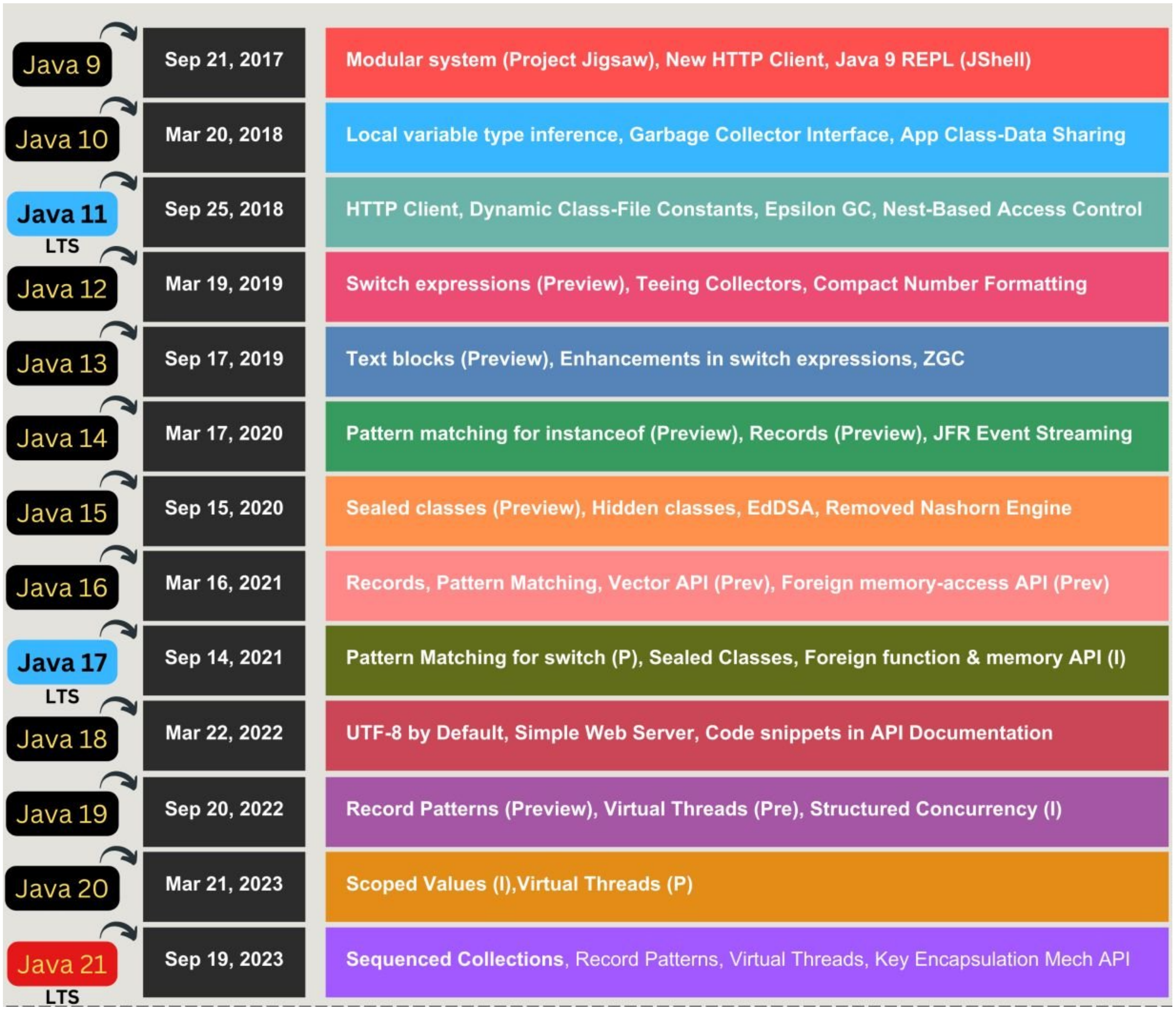
De Java 8…
à Java 21



Java 9	Sep 21, 2017	Modular system (Project Jigsaw), New HTTP Client, Java 9 REPL (JShell)
Java 10	Mar 20, 2018	Local variable type inference, Garbage Collector Interface, App Class-Data Sharing
Java 11 LTS	Sep 25, 2018	HTTP Client, Dynamic Class-File Constants, Epsilon GC, Nest-Based Access Control
Java 12	Mar 19, 2019	Switch expressions (Preview), Teeing Collectors, Compact Number Formatting
Java 13	Sep 17, 2019	Text blocks (Preview), Enhancements in switch expressions, ZGC
Java 14	Mar 17, 2020	Pattern matching for instanceof (Preview), Records (Preview), JFR Event Streaming
Java 15	Sep 15, 2020	Sealed classes (Preview), Hidden classes, EdDSA, Removed Nashorn Engine
Java 16	Mar 16, 2021	Records, Pattern Matching, Vector API (Prev), Foreign memory-access API (Prev)
Java 17 LTS	Sep 14, 2021	Pattern Matching for switch (P), Sealed Classes, Foreign function & memory API (I)
Java 18	Mar 22, 2022	UTF-8 by Default, Simple Web Server, Code snippets in API Documentation
Java 19	Sep 20, 2022	Record Patterns (Preview), Virtual Threads (Pre), Structured Concurrency (I)
Java 20	Mar 21, 2023	Scoped Values (I),Virtual Threads (P)
Java 21 LTS	Sep 19, 2023	**Sequenced Collections**, Record Patterns, Virtual Threads, Key Encapsulation Mech API

Évolution

Langage

Outils: `jshell`, `jlink`, `jdeps`, `jpackage`, `java` *source*, `javadoc`

Bibliothèques: HTTP client, Collection factories, Unix-domain sockets, Stack walker, Deserialization filtering, Pseudo-RNG, SHA-3, TLS 1.3...

JVM: Garbage Collectors: G1, ZGC, AArch64 support, Docker awareness, Class Data Sharing by default, Helpful Null Pointer Exceptions...

Optional

```java
Optional.of(new Random().nextInt(10))
    .filter(i -> i % 2 == 0)
    .map(i -> "number is even: " + i)
    .ifPresent(System.out::println);
```

Optional: monade qui enveloppe une référence qui peut ou non être nulle

Lambda Expressions et Stream API

```java
public class IterationOld {
    public static List<Car> findCarsOldWay(List<Car> cars) {
        List<Car> selectedCars = new ArrayList<>();
        for (Car car : cars) {
            if (car.kilometers < 50000) {
                selectedCars.add(car);
            }
        }
        return selectedCars;
    }
}
```

```java
public class LambdaExpressions {
    public static List<Car> findCarsUsingLambda(List<Car> cars) {
        return cars.stream().filter(car -> car.kilometers < 50000)
                .toList();
    }
}
```

Lambda Expressions et Stream API (cont.)

```java
public class withoutMethodReference {
    List<String> withoutMethodReference =
         cars.stream().map(car -> car.toString())
                    .toList();
}
```

```java
public class withMethodReference {
    List<String> methodReference =
         cars.stream().map(Car::toString)
                  .toList();
}
```

Local variable inference

```
public record Person(String name, String lastname) { }

public void varTypes() {
    var Roland = new Person("Roland", "Deschain");
    var Susan = new Person("Susan", "Delgado");
    var Eddie = new Person("Eddie", "Dean");
    var Detta = new Person("Detta", "Walker");
    var Jake = new Person("Jake", "Chambers");

    var persons = List.of(Roland, Susan, Eddie, Detta, Jake);

    for (var person : persons) {
        System.out.println(person.name + " - " + person.lastname);
    }
}
```

Switch expression

```java
private static void oldStyleWithBreak(Fruit fruit) {
   switch (fruit) {
      case APPLE, PEAR:
         System.out.println("Common fruit");
         break;
      case ORANGE, AVOCADO:
         System.out.println("Exotic fruit");
         break;
      default:
         System.out.println("Undefined fruit");
   }
}
```

Switch expression (cont.)

```java
private static void withSwitchExpression(Fruit fruit) {
   switch (fruit) {
          case APPLE, PEAR -> System.out.println("Common fruit");
          case ORANGE, AVOCADO -> System.out.println("Exotic fruit");
          default -> System.out.println("Undefined fruit");
   }
}
```

```java
private static void withReturnValueEvenShorter(Fruit fruit) {
      System.out.println(switch (fruit) {
              case APPLE, PEAR -> "Common fruit";
              case ORANGE, AVOCADO -> "Exotic fruit";
              default -> "Undefined fruit";
      });
}
```

Switch expression (cont.)

```java
private static void withYield(Fruit fruit) {
   String text = switch (fruit) {
          case APPLE, PEAR -> {
              System.out.println("the given fruit was: " + fruit);
              yield "Common fruit";
          }
          case ORANGE, AVOCADO -> "Exotic fruit";
          default -> "Undefined fruit";
   };
   System.out.println(text);
}
```

Text blocks

```
String text = "{\n" +
              "  \"name\": \"John Doe\",\n" +
              "  \"age\": 45,\n" +
             "}";
System.out.println(text);
```

```
String text = """
        {
          "name": "John Doe",
          "age": 45,
       }
        """;
System.out.println(text);
```

```
String text = """
        {
          "name": "John Doe",
          "age": 45,
       }
               """;
System.out.println(text);
```

instanceof pattern matching

```
public class PatternMatching {
    public static void main(String[] args) {
        Car car = new Car("Kia", "Gas", 10000l, 2021l);
        Bicycle bicycle = new Bicycle("Greyp", "Electricity", "Mountain", 21l);
        System.out.println("Car:" + price(car));
        System.out.println("Bycicle: " + price(bicycle));
    }
   public static double price(Vehicle v) {
        if (v instanceof Car c) {
            return 10000 - c.kilometers * 0.01 -
                    (Calendar.getInstance().get(Calendar.YEAR) - c.year) * 100;
        } else if (v instanceof Bicycle b) {
            return 1000 + b.wheelSize * 10;
        } else throw new IllegalArgumentException();
    }
}
```

Record

Tuple **immuable**, avec un nom, et un état

- final classe
- final attributs
- toString, hashCode, equals
- accesseurs

Record

Intéressant

- pour modéliser des éléments tels que des classes du domaine métier (à persister via ORM), ou des DTO
- pour stocker temporairement des données (par ex. désérialisation JSON) et les sécuriser (cf. cours Sécurité)
- pour conserver les données pendant un certain temps et pour éviter d'écrire du code "boilerplate" (getter/setter,…)
- pour conserver plusieurs valeurs de retour d'une méthode, des jointures de flux, des clés composées et des structures de données telles que des nœuds d'arbres

Record (cont.)

```java
public class Raisin {
    private final Color couleur;
    private final int nbGrains;
    public Raisin(Color couleur, int nbGrains) {
        this.couleur = couleur;
        this.nbGrains = nbGrains;
    }
    ... // code "inutile"
}
```

```java
    public Color getCouleur() {
        return couleur;
    }
    public int getNbGrains() {
        return nbGrains;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Raisin that = (Raisin) o;
        return nbGrains == that.nbGrains && couleur.equals(that.color);
```

Record (cont.)

```
public record RaisinRecord(Color couleur, int nbGrains) { }
```

```
record RaisinRecord(Color couleur, int nbGrains) {
         RaisinRecord {
             System.out.println("Paramètre Couleur=" + couleur +
               ", Attribut couleur=" + this.couleur());
             System.out.println("Paramètre nbGrains=" + nbGrains +
               ", Field nbGrains=" + this.nbGrains());
             if (couleur == null) {
                 throw new IllegalArgumentException("Couleur ne doit pas
                   être nulle");
             }
}
```

Record (cont.)

Record de base

```
record User(UUID id, String name) { }
```

Valeur par défaut

```
record User(UUID id, String name) {
  /** Création d'un nouvel user avec un ID par défaut */
  public User(String name) {
     this(UUID.randomUUID(), name);
  }
}
```

```
var user = new User("John");
UUID generated = user.id();
```

Record (cont.)

Validation

```
record User(UUID id, String name) {
   public User {
      if(name.isBlank()) {
         throw new IllegalArgumentException("name cannot be empty.");
      }
   }
}
```

Normalisation

```
record User(UUID id, String name) {
   User {
      name = name.trim();
   }
}
```

```
User user = new User(UUID.randomUUID(), "John \n");
String name = user.name();      // "John"
```

Record (cont.)

Modification

```
record User(UUID id, String name) {
    User withName(String name) {
        return new User(id, name);
    }
}
```

```
var user = new User(UUID.randomUUID(), "John");
var renamed = user.withName("John Doe");
```

Non null

```
record User(UUID id, String name) {
    User {
        Objects.requireNonNull(id, "id cannot be null");
        Objects.requireNonNull(name, "name cannot be null");
    }
}
```

Record (cont.)

Données dérivées

```java
record User(String firstName, String lastName) {
    public String fullName() {
        return String.format("%s %s", firstName, lastName);
    }
}
```

```java
var user = new User("John", "Doe");
var fullName = User.fullName();        //"John Doe"
```

Sealed classes

```java
public abstract class Fruit {
}
public final class Apple extends Fruit {
}
public final class Pear extends Fruit {
}
```

```java
private static void problemSpace() {
      Apple apple = new Apple();
      Pear pear = new Pear();
      Fruit fruit = apple;
      class Avocado extends Fruit {};
}
```

Sealed classes (cont.)

```
public abstract sealed class FruitSealed permits AppleSealed, PearSealed {
}
public non-sealed class AppleSealed extends FruitSealed {
}
public final class PearSealed extends FruitSealed {
}
```

```
private static void sealedClasses() {
      AppleSealed apple = new AppleSealed();
      PearSealed pear = new PearSealed();
      FruitSealed fruit = apple;
      class Avocado extends AppleSealed {};
}
```

Sealed classes (cont.)

Héritage contrôlé/Sous-classes limitées : le mot-clé `permits` spécifie une liste de classes qui peuvent être des sous-classes directes de la classe scellée

Ouverte et non scellée : les classes scellées peuvent également permettre aux classes qui les étendent d'être ouvertes ou non scellées, ce qui offre une plus grande souplesse dans la conception de la hiérarchie des classes

Finale par défaut : les classes scellées sont implicitement finales, ce qui signifie qu'elles ne peuvent pas être directement instanciées ou sous-classées, sauf autorisation explicite

Cas d'utilisation : les classes scellées sont utiles pour modéliser des hiérarchies fermées où l'on souhaite limiter le nombre de sous-classes, ce qui permet de mieux contrôler la conception et le comportement de la hiérarchie des classes

Null Pointer Exceptions

```java
public static void main(String[] args) {
      HashMap<String, Raisin> grappe = new HashMap<>();
      grappe.put("grappe1", new Raisin(Color.BLACK, 2));
      grappe.put("grappe2", new Raisin(Color.WHITE, 4));
      grappe.put("grappe3", null);
      var couleur = ((Raisin) grappe.get("grappe3")).getColor();
}
```

```
Exception in thread "main" java.lang.NullPointerException
       at Test.main(Test.java.java:9)
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"Raisin.color()" because the return value of "java.util.HashMap.get(Object)" is null
       at Test.main(Test.java:9)
```

Compact Number Formatting

```
public static void main(String[] args) {
        var fmt = NumberFormat
             .getCompactNumberInstance(Locale.ENGLISH, NumberFormat.Style.SHORT);
        System.out.println(fmt.format(1000));
        System.out.println(fmt.format(100000));
        System.out.println(fmt.format(1000000));

        fmt = NumberFormat
             .getCompactNumberInstance(Locale.ENGLISH, NumberFormat.Style.LONG);
        System.out.println(fmt.format(1000));
        System.out.println(fmt.format(100000));
        System.out.println(fmt.format(1000000));

        fmt = NumberFormat
             .getCompactNumberInstance(Locale.forLanguageTag("FR"),
                                       NumberFormat.Style.LONG);
        System.out.println(fmt.format(1000));
        System.out.println(fmt.format(100000));
        System.out.println(fmt.format(1000000));
}
```

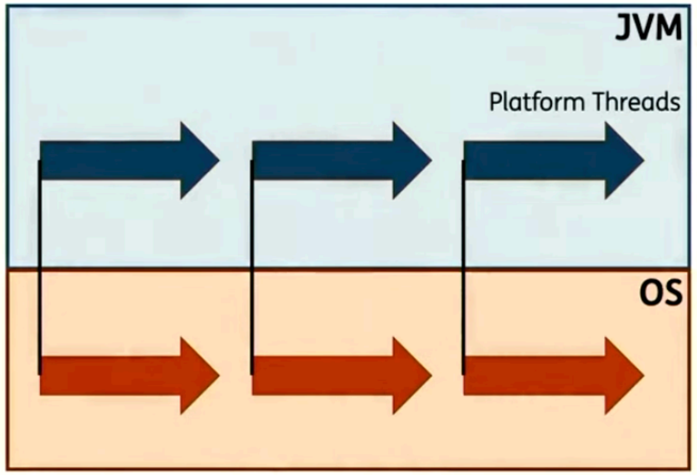
Date Formatting

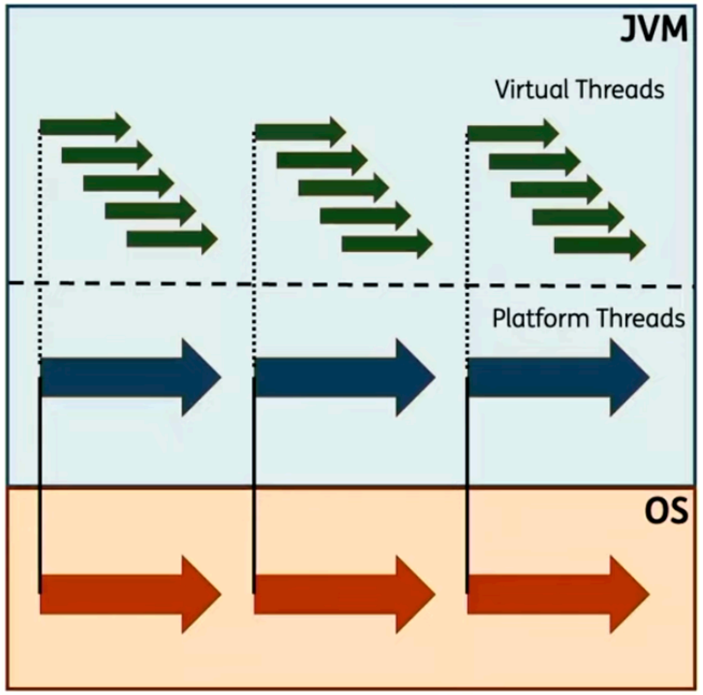
```java
public static void main(String[] args) {
        System.out.println("""
                ----- English -----""");
        var dtf = DateTimeFormatter.ofPattern("B")
                        .withLocale(Locale.forLanguageTag("ENG"));
        System.out.println(dtf.format(LocalTime.of(8, 0)));
        System.out.println(dtf.format(LocalTime.of(13, 0)));
        System.out.println(dtf.format(LocalTime.of(20, 0)));
        System.out.println(dtf.format(LocalTime.of(23, 0)));
        System.out.println(dtf.format(LocalTime.of(0, 0)));

        System.out.println("""
                ----- Français -----""");
        dtf = DateTimeFormatter.ofPattern("B")
                    .withLocale(Locale.forLanguageTag("FR"));
        System.out.println(dtf.format(LocalTime.of(8, 0)));
        System.out.println(dtf.format(LocalTime.of(13, 0)));
        System.out.println(dtf.format(LocalTime.of(20, 0)));
        System.out.println(dtf.format(LocalTime.of(0, 0)));
        System.out.println(dtf.format(LocalTime.of(1, 0)));
}
```

Record patterns

```java
    public static void instanceofExample() {
        var point = new Point(10, 10);
        var coloredPoint = new ColoredPoint(point, "blue");
        Object obj = coloredPoint;

        if (obj instanceof ColoredPoint cp) {
            System.out.println("obj is a ColoredPoint: " + cp);
        }

        if (obj instanceof Point p) {
            System.out.println("obj is a point: " + p);
        } else {
            System.out.println("obj is not a point");
        }

        if (obj instanceof ColoredPoint(Point(int x, var y), String color)) {
            System.out.printf("Point [%d,%d] has color %s%n", x, y, color);
        }
    }

record Point(int x, int y) { }
record ColoredPoint(Point p, String color) { }
```

Virtual threads

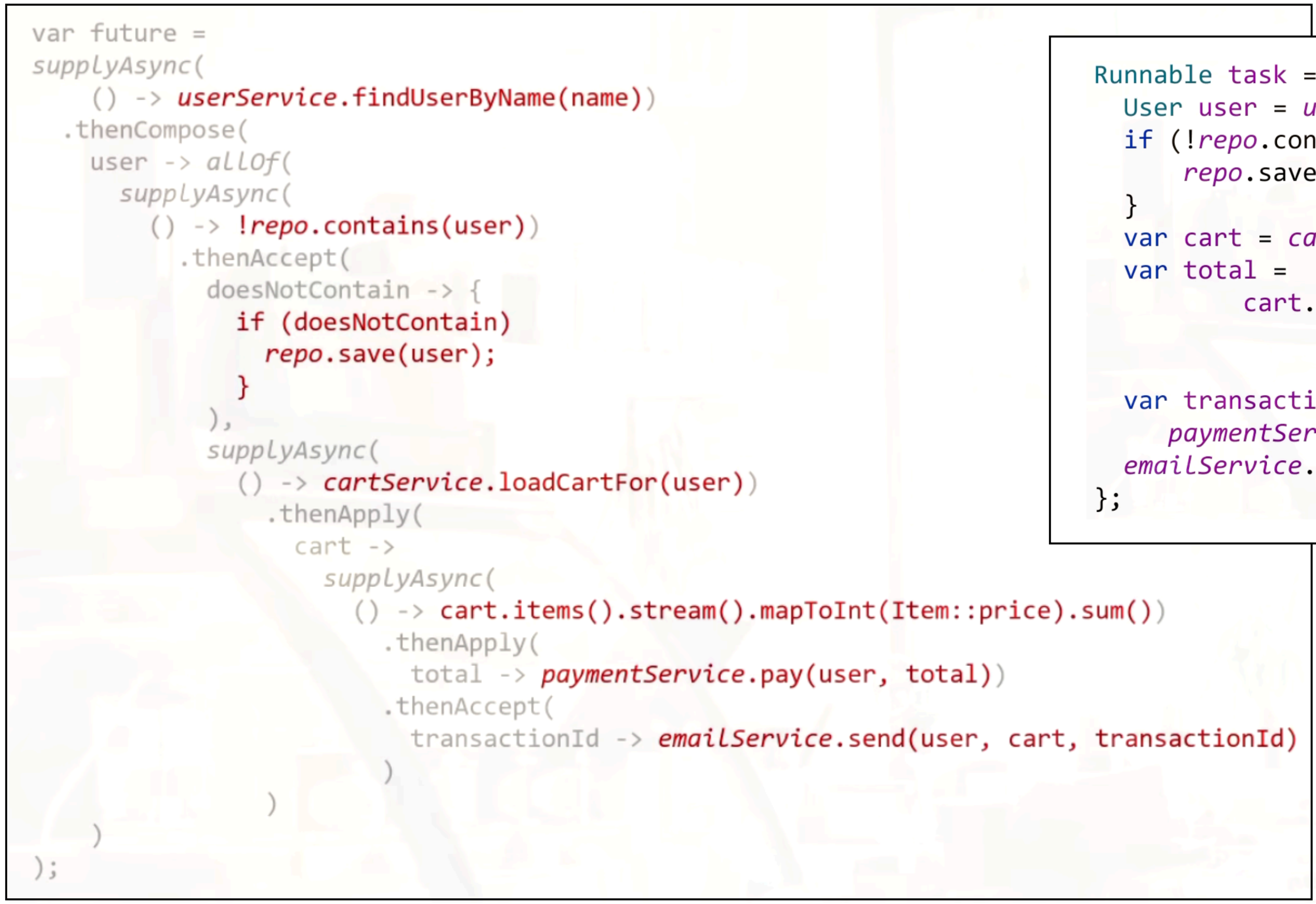




Virtual threads (cont.)

```java
public static void platformThread() {
      Thread.ofPlatform().start(() -> System.out.println(Thread.currentThread()));
}

public static void virtualThread() {
      Thread.ofVirtual().start(() -> System.out.println(Thread.currentThread()));
}

public static void virtualThreadWithExecutor() {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        executor.submit(() -> System.out.println(Thread.currentThread()));
        executor.submit(() -> System.out.println(Thread.currentThread()));
        executor.submit(() -> System.out.println(Thread.currentThread()));
    }
}
```

Structured concurrency



```
var future =
supplyAsync(
    () -> userService.findUserByName(name))
  .thenCompose(
    user -> allOf(
      supplyAsync(
        () -> !repo.contains(user))
          .thenAccept(
            doesNotContain -> {
              if (doesNotContain)
                repo.save(user);
              }
            ),
            supplyAsync(
              () -> cartService.loadCartFor(user))
                .thenApply(
                  cart ->
                    supplyAsync(
                      () -> cart.items().stream().mapToInt(Item::price).sum())
                        .thenApply(
                          total -> paymentService.pay(user, total))
                        .thenAccept(
                          transactionId -> emailService.send(user, cart, transactionId)
                        )
                  )
      )
);
```

```
Runnable task = () -> {
  User user = userService.findUserByName(name);
  if (!repo.contains(user)) {
      repo.save(user);
  }
  var cart = cartService.loadCartFor(user);
  var total =
          cart.items().stream()
                  .mapToInt(Item::price)
                  .sum();
  var transactionId =
     paymentService.pay(user, total);
  emailService.send(user, cart, transactionId);
};
```

Structured concurrency

```java
public static void getWeather() {
    var future1 = CompletableFuture.supplyAsync(() -> WeatherUtil.getWeatherFromSource1("Amsterdam"));
    var future2 = CompletableFuture.supplyAsync(() -> WeatherUtil.getWeatherFromSource2("Amsterdam"));
    var future3 = CompletableFuture.supplyAsync(() -> WeatherUtil.getWeatherFromSource3("Amsterdam"));

    CompletableFuture.anyOf(future1, future2, future3)
            .exceptionally(th -> {
                throw new RuntimeException(th);
            })
            .thenAccept(weather -> System.out.println("Weather: " + weather))
            .join();
}
```

```java
public static void getWeather() {
    try (var scope = new StructuredTaskScope.ShutdownOnSuccess<WeatherUtil.Weather>()) {
        scope.fork(() -> WeatherUtil.getWeatherFromSource1("Amsterdam"));
        scope.fork(() -> WeatherUtil.getWeatherFromSource2("Amsterdam"));
        scope.fork(() -> WeatherUtil.getWeatherFromSource3("Amsterdam"));
        var weather = scope.join().result();
        System.out.println(weather);
    } catch (InterruptedException | ExecutionException e) { throw new RuntimeException(e); }
}
```

Sequence collections

```
ArrayList<Integer> arrayList = new ArrayList<>();

arrayList.add(1);                               // [1]
arrayList.addFirst(0);                          // [0, 1]
arrayList.addLast(2);                           // [0, 1, 2]

Integer firstElement = arrayList.getFirst();    // 0
Integer lastElement = arrayList.getLast();      // 2

List<Integer> reversed = arrayList.reversed();
System.out.println(reversed);                   // Prints [2, 1, 0]

arrayList.add(3);
System.out.println( arrayList );                // [0, 1, 2, 3]
System.out.println( arrayList.reversed() );     // [3, 2, 1, 0]
```

Synthèse

Maintenabilité: records, sealed classes, null pointer exceptions, switch

Concision: lambda expressions, local variable inference, switch, instance of

Simplicité: text blocks, compact number & date formatting, sequence collections, lambda expressions