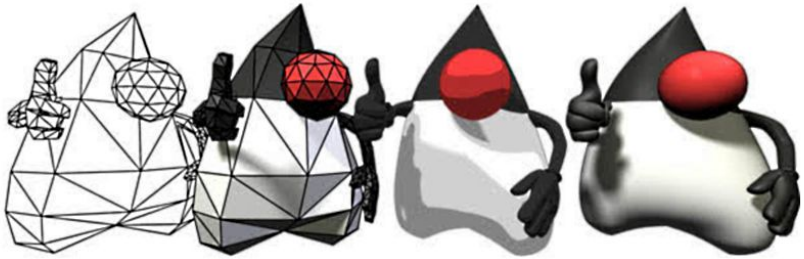
Luxoft
A DXC Technology Company



New Features in Java

Rahman Usta
rahman.usta@dxc.com

/ustarahman

/rahmanusta

New Features in Java

- **JEP 286: Local-Variable Type Inference (Java 10)**
- **JEP 358: Helpful NullPointerExceptions (Java 14)**
- **JEP 361: Switch Expressions (Java 14)**
- **JEP 378: Text Blocks (Java 15)**
- **JEP 394: Pattern Matching for instanceof (Java 16)**
- **JEP 395: Records (Java 16)**
- **JEP 409: Sealed Classes (Java 17)**
- **JEP 406: Pattern Matching for switch (Preview) (Java 17)**

See the other features:
https://openjdk.java.net/projects/jdk/<version>/

New release every 6 months

Java 10, Java 11 (**LTS**) => 2018 (March & September)

Java 12, Java 13 => 2019 (March & September)

Java 14, Java 15 => 2020 (March & September)

Java 16, Java 17 (**LTS**) => 2021 (March & September)

- **LTS** -> Long Term Support
- New LTS every 3 years

Java SE Support Roadmap

Oracle Java SE Support Roadmap*†			
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**
7	July 2011	July 2019	July 2022*****
8**	March 2014	March 2022	December 2030*****
9 (non-LTS)	September 2017	March 2018	Not Available
10 (non-LTS)	March 2018	September 2018	Not Available
11 (LTS)	September 2018	September 2023	September 2026
12 (non-LTS)	March 2019	September 2019	Not Available
13 (non-LTS)	September 2019	March 2020	Not Available
14 (non-LTS)	March 2020	September 2020	Not Available
15 (non-LTS)	September 2020	March 2021	Not Available
16 (non-LTS)	March 2021	September 2021	Not Available
17 (LTS)	September 2021***	September 2026****	September 2029****

Preview Features

- Not a standard yet
- Ready to try, review and feedback
- May be changed, even removed
- Can be staged, first preview, second preview etc.

Enable preview features, disabled by default

--enable-preview

JEP 286: Local-Variable Type Inference[10]

Local-Variable Type Inference

```
public void someMethod() {

   List<Integer> numbers1 = new ArrayList<>();
   Stream<Integer> stream1 = numbers1.stream();

   var numbers2 = new ArrayList<>();
   var stream2 = numbers2.stream();

}
```

JEP 358: Helpful NullPointerExceptions[14]

Pre Helpful NullPointerExceptions

```
Person person = new Person();
person.address = new Address();

String toUpperCase = person.address.street.toUpperCase();
System.out.println(toUpperCase);
```

```
Exception in thread "main" java.lang.NullPointerException
        at java14.edu/com.kodedu.NullPointerException.main(NullPointerException.java:10)
```

Helpful NullPointerExceptions

```
Person person = new Person();
person.address = new Address();

String toUpperCase = person.address.street.toUpperCase();
System.out.println(toUpperCase);
```

New flag!

`-XX:{+|-}ShowCodeDetailsInExceptionMessages`

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.toUpperCase()"
because "person.address.street" is null
        at java14.edu/com.kodedu.NullPointerException.main(NullPointerException.java:10)
```

JEP 361: Switch Expressions (Standard[14])

Pre Switch Expressions

```
int speedLimit;
switch (vehicleType) {
   case BIKE:
   case SCOOTER:
      speedLimit = 40;
      break;
   case MOTORBIKE:
   case AUTOMOBILE:
      speedLimit = 140;
      break;
   case TRUCK:
      speedLimit = 80;
      break;
   default:
      throw new IllegalStateException("No case found for: " + vehicleType);
}

System.out.println("Speed limit: " + speedLimit);
```

Switch Expressions

```
VehicleType vehicleType = VehicleType.AUTOMOBILE;

int speedLimit = switch (vehicleType) {
   case BIKE, SCOOTER -> 40;
   case MOTORBIKE, AUTOMOBILE -> 140;
   case TRUCK -> 80;
   default -> throw new IllegalStateException("No case found for: " + vehicleType);
};

System.out.println("Speed limit: " + speedLimit);
```

All enum cases have to be covered in switch block!

Switch Expressions

```
int speedLimit = getSpeedLimit(VehicleType.AUTOMOBILE);
System.out.println("Speed limit: " + speedLimit);

private static int getSpeedLimit(VehicleType vehicleType) {
   return switch (vehicleType) {
       case BIKE, SCOOTER -> 40;
       case MOTORBIKE, AUTOMOBILE -> 140;
       case TRUCK -> 80;
   };
}
```

Switch Expressions : yield

```
VehicleType vehicleType = VehicleType.TRUCK;

int speedLimit = switch (vehicleType) {
   case BIKE, SCOOTER -> 40;
   case MOTORBIKE, AUTOMOBILE -> 140;
   case TRUCK -> {
      int randomSpeed = ThreadLocalRandom.current().nextInt(70, 80);
      yield randomSpeed;
   }
};

System.out.println("Speed limit: " + speedLimit);
```

JEP 378: Text Blocks (Standard[15])

Pre Text Blocks

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

Text Blocks

```
String html = """
              <html>
                  <body>
                      <p>Hello, world</p>
                  </body>
              </html>
              """;
```

```
<html>⏎
••••<body>⏎
••••••••<p>Hello,•world</p>⏎
••••</body>⏎
</html>⏎
```

Text Blocks

```
// ""
var text = """
           """;
```

Line terminator required after opening delimiter

```
// illegal text block start
var text = """""";
```

Text Blocks : Indentation

```
String html = """
                  <html>
                     <body>
                       <p>Hello, world</p>
                     </body>
                  </html>
              """;
```

```
····<html>⏎
·······<body>⏎
·········<p>Hello,•world</p>⏎
·······</body>⏎
····</html>⏎
```

Text Blocks : Indentation

```
String html = """
                    <html>
                       <body>
                         <p>Hello, world</p>
                       </body>
                    </html>
                    """;
```

```
<html>⏎
•••<body>⏎
•••••<p>Hello,•world</p>⏎
•••</body>⏎
</html>⏎
```

Text Blocks : Espace line terminator

```
String html = """
              <html> \
                 <body> \
                   <p>Hello, world</p> \
                 </body> \
              </html> \
              """;
```

```
<html>••••<body>••••••<p>Hello,•world</p>•••</body>•</html>•
```

Text Blocks : Single space character

```
String html = """
        <html>                    \s
           <body>                 \s
             <p>Hello, world</p>\s
           </body>                \s
        </html>                   \s
        """;
```

```
<html>•••••••••••••••••••
•••<body>•••••••••••••••
•••••<p>Hello,•world</p>•
•••</body>•••••••••••••••
</html>•••••••••••••••••••
```

Text Blocks : String#formatted

```
String html = """
              <html>
                 <body>
                   <p>%s, %s</p>
                 </body>
              </html>
              """;
html.formatted("Hello", "World");
```

```
<html>⏎
•••<body>⏎
•••••<p>Hello,•World</p>⏎
•••</body>⏎
</html>⏎
```

JEP 394: Pattern Matching for instanceof (Standard[16])

Pre Pattern Matching

```
Object obj = "Hello world!";

if (obj instanceof String) {
   String s = (String) obj;
   System.out.println("String: " + s);
}
```

Pattern Matching

```
if (obj instanceof String s) {
    System.out.println("String: " + s);
}
```

```
// cannot resolve symbol 's'
if (obj instanceof String s || !s.isBlank()) {
    System.out.println("String: " + s);
}
```

```
// legal usage
if (obj instanceof String s && !s.isBlank()) {
    System.out.println("String: " + s);
}
```

JEP 395: Records (Standard[16])

Pre Records

```
public class Point {
    private int x;
    private int y;

    // constructor
    // setters & getters
    // equals & hashcode
    // toString

}
```

```
public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

```
public int getX() {
    return x;
}

public void setX(int x) {
    this.x = x;
}

public int getY() {
    return y;
}

public void setY(int y) {
    this.y = y;
}
```

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Point point = (Point) o;
    return x == point.x &&
            y == point.y;
}

@Override
public int hashCode() {
    return Objects.hash(x, y);
}
```

```
@Override
public String toString() {
    return "Point{" +
            "x=" + x +
            ", y=" + y +
            '}';
}
```

Records

```
record Point(int x, int y){ }
```

- 1 canonical constructor
- Fields are final
- No setter but getters -> point.x() , point.y()
  - Records are shallowly immutable!
- Default implementation of hashCode and equals
- A standard toString implementation “Point[x=1, y=2]”
- Default characteristic can be overridden
- Record classes can’t extend/be extended
- Can implement interfaces
- Can be declared locally

JEP 409: Sealed Classes (Standard[17])

Sealed Classes

too **restrictive**

A **final** class

cannot have any subclass(es)

restrictive as API developer's desire

A **sealed** class

defines what are the sum of subtypes.

too **permissive**

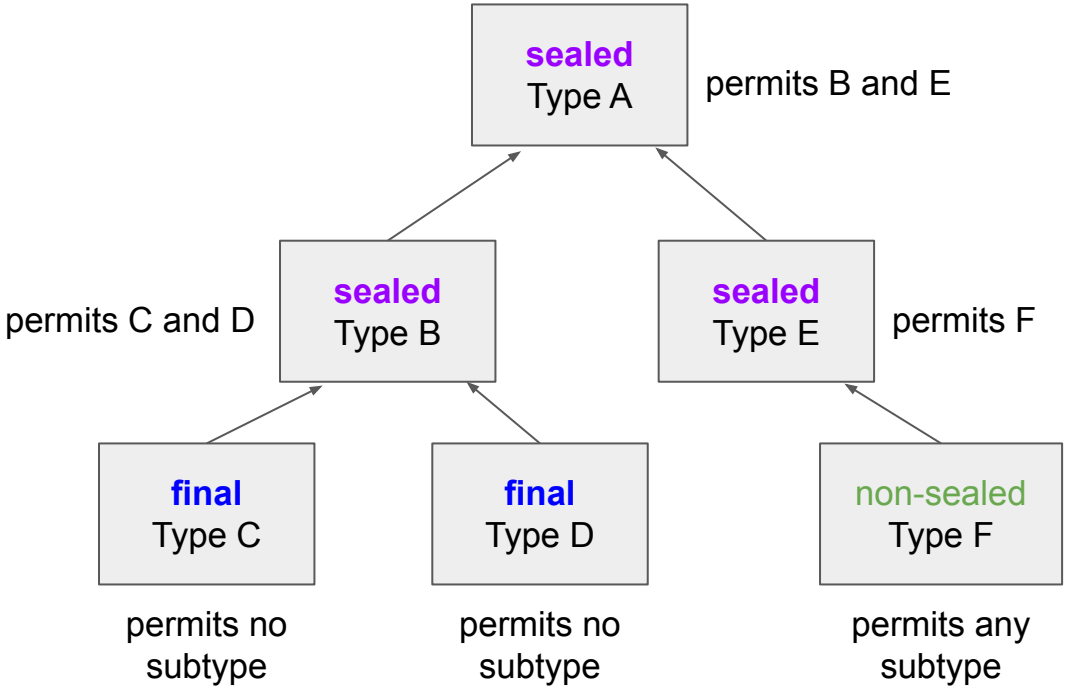
A **non-final** class

may have subclass(es)

Sealed Classes

```
public sealed class Shape permits Square, Circle {

}

public final class Square extends Shape {

}

public final class Circle extends Shape {

}
```

Sealed Classes : Exhaustive



Algebraic types: Records + Sealed Classes

```
sealed interface Expr permits  ConstantExpr, NegExpr, PlusExpr, TimesExpr   { }

record ConstantExpr(int i) implements Expr { }
record PlusExpr(Expr a, Expr b) implements Expr { }
record TimesExpr(Expr a, Expr b) implements Expr { }
record NegExpr(Expr e) implements Expr { }
```

- Records
  - Defines product types
- Sealed classes
  - Defines sum of types

```
int calculate(Expr e) {
        return switch (e) {
                case ConstantExpr(var i) -> i;
                case PlusExpr(var a, var b) -> calculate(a) + calculate(b);
                case TimesExpr(var a, var b) -> calculate(a) * calculate(b);
                case NegExpr(var e) -> -calculate(e);
                // no default needed, Expr is sealed
  }
}
```

Stream API changes

- Stream#toList
  - An alternative to Stream#collect(Collectors#toList())
  - Creates an immutable list
- Stream#mapMulti
  - N input -> M output

Sample code: https://github.com/rahmanusta/java16-edu/tree/master/src/main/java/com/kodedu/stream

JEP 406: Pattern Matching for switch (Preview[17])

Pattern matching for switch

```
public String formatterPatternSwitch(Object o) {
   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();
   };
}
```

Guarded patterns

```
public void test(Object o) {
    switch (o) {
      case String s && (s.length() == 1) -> ...
      case String s   -> ...
      ...
    }
}
```

Try Java 17

Open-source builds

- https://jdk.java.net/17

Online Java Shell

- https://tryjshell.org/

Code samples

- https://github.com/rahmanusta/java-next-edu

Thank you!