ORACLE®

Java™

ORACLE®

Future of Java

Post-JDK 9 Candidate Features

Jan Lahoda
Java compiler developer
Java Product Group, Oracle
September, 2017

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Overview

- Java language and platform evolution goals:
  - Make it easier to build and maintain reliable programs
  - Keep migration compatibility
- Reading is more important than writing
- Many enhancements done over the years
- Some possible future enhancements noted here
  - Some may happen sooner, some later, some never
  - Anything can change, no specific timeline

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Schedule

- More frequent (feature) releases
- Proposed:
  - Feature releases every 6 months (March, September)
  - Long-term support releases every 3 years
- More in the proposal:
  - https://mreinhold.org/blog/forward-faster

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Improving type inference

- Java is strictly typed, no change planned
- Explicit types not needed in many cases, though:
  - `Set<String> ns = Collections.<String>emptySet();`
  - `Set<String> ns = new HashSet<String>();`
  - `ns.removeIf( (String s) -> s.isEmpty());`
- Common property: does not affect API
  - Contracts should be explicit

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Improving type inference

- Next opportunity: type inference for **local** variables

```
URL url = new URL("http://www.oracle.com/");
URLConnection conn = url.openConnection();
Reader reader = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
```

- Becomes:

```
var url = new URL("http://www.oracle.com/");
var conn = url.openConnection();
var reader = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Taming Boilerplate - Data Classes

- Some boilerplate has been avoided (e.g. lambdas)
- More remains, e.g. “mandatory” methods in domain objects:

```
public class Point {
    public final int x;
    public final int y;
    public Point(int x, int y) { … }

    @Override
    public int hashCode() { … }

    @Override
    public boolean equals(Object obj) { … }

    @Override
    public String toString() { … }

}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Taming Boilerplate - Data Classes

- IDEs can generate these methods
- Need to be maintained, read, etc.
- How about:
  `public class Point (int x, int y) {}`
- Constructor, equals, hashCode, toString autogenerated
- +further important methods could be as well

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- Motivation

- A common code:

```
if (obj instanceof Integer) {
    Integer i = (Integer) obj;
    int v = i.intValue();
    System.err.println(“Integer: “ + v);
}
```

- Check a condition, cast and retrieve attribute(s)
- Verbose and error-prone

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- matches with bind

- How about:

```
if (obj matches Integer i) {
    int v = i.intValue();
    System.err.println(“Integer: “ + v);
}
```

- matches combines instanceof and variable binding (+more)
- Much clearer, safer

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- matches with nested patterns

- Or even:

```
if (obj matches Integer(int v)) {
    System.err.println(“Integer: “ + v);
}
```

- “`int v`” is a nested pattern – looking “inside” the object (could use a new “mandatory” method, btw)
- Patterns can nest as deep as needed
  - `class Line(Point start, Point end) {}`
  - `Line(Point(int sX,int sY),Point(int eX,int eY))`

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- other patterns

- Type in (nested) pattern can be inferred:

```
   Line(Point start, Point end)
=> Line(var start, var end)
```

- Or unimportant elements ignored:

```
Line(var start, _)
```

- Constants can be patterns too:

```
x matches 42
x matches Line(Point(0, 0), _)
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- switch

- Switch statement fairly limited:
  - Only accepts int, enum and String

- “multi-arm if” - could use patterns as well?

```
switch (expr) {
  case Integer i: println(“Integer: “ + i); break;
  case Double  d: println(“Double: “ + d); break;
  case Point(int x, var y):
    println(“Point: “ + x + “, “ + y); break;
}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- switch

- Switch expression of any type
- Also supports “`case null:`”
- Migration aided using constant patterns:

```
switch (expr) {
  case 42: println(“42!”); break;
  case Integer i: println(“Integer: “ + i); break;
}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- Exhaustive switch

- String text;

```
switch (expr) {
   case Integer i: text = “Integer: “ + i; break;
   case Double  d: text = “Double: “ + d; break;
   default: text = “Unknown”; break;
}
```

- Relies on definite assignment (DA)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- Exhaustive switch

- How about:
- String text =

```
  switch (expr) {
    case Integer i -> “Integer: “ + i;
    case Double  d -> “Double: “ + d;
    default -> “Unknown”;
  }
```

- More obvious all variants covered (checked)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Pattern Matching

- Conclusion

- Patterns:
  - Constant patterns
  - Type test patterns
  - Destructuring (nested) patterns
  - var patterns
  - '_' (anything)
  - 'case null:'

- Uses:
  - Matches expression
  - Pattern based switch
  - Expression switch
- Likely to be done in phases over several releases
- Currently prototyped:
  - constant and type test patterns

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Valhalla

- Memory access (cache miss) is slow – dereferences costly
- Consider:
  ```
  Point[] pArr = …
  pArr[0].x + pArr[1].x + ...
  ```
- The array is an array of pointers to the actual data:
  - `[0] → [x0, y0]`
  - `[1] → [x1, y1]`
  - `[2] → [x2, y2]`

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Valhalla

- For “int[]” - ints are inlined in the array:
  - `[x0, x1, x2, ….]`
- How about inlining custom classes?
  - `[[x0, y0], [x1, y1], [x2, y2]]`
- But without compromising readability and maintainability
- => value classes

Valhalla

- Value Classes
- “codes like a class, works like an int”
- Do not have identity, only value
- Their values inlined in arrays, enclosing objects:

  ```
  Line { Point start; Point end; }
  =>
  Line { start_x; start_y; end_x; end_y; }
  ```
- “user-defined primitive”

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Valhalla

- Value Classes
- Significant changes needed for full support
- Currently works on “minimal value types” prototype
- To evaluate and experiment without significant language changes

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Conclusion

- Many new features under investigation:
  - Improved type inference (“local variable type inference”)
  - Data classes
  - Pattern matching
  - Value classes
  - (and many more)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Conclusion

- Continued
- Everything is a subject to change
  - Things may or may not happen
  - No specific timeline/release
  - Details likely to change

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Q & A

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Integrated Cloud

Applications & Platform Services

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Java™

ORACLE®

ORACLE®