



Istanbul
Java User Group

<https://kommunity.com/istanbul-jug>

Java 16 Features

Rahman Usta

rahmanusta@kodedu.com



/ustarahman



/rahmanusta

Rahman Usta



- Senior Developer and Software Trainer in Luxoft Poland
- Based in Istanbul
- Book author and blogger
- Java champion
- Open source contributor
 - <https://github.com/rahmanusta>

Java 16 Features

- JEP 357 - Migrate from Mercurial to Git
- JEP 369 - Migrate to GitHub
- JEP 375 - Pattern Matching for instanceof (Standard)
- JEP 384 - Records Component (Standard)
- JEP 360 - Sealed Classes (Second Preview)
- JEP 389 - Foreign Linker API (Incubator)
- JEP 380 - Unix-Domain Socket Channels (Standard)

See all features:

<https://openjdk.java.net/projects/jdk/16/>

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 => 2021 (March)

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

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 357: Migrate from Mercurial to Git

JEP 369: Migrate to GitHub

<https://github.com/openjdk/>

Reasons to Migrate GIT

GIT is kinda standard among version control systems

- Size on Disk
 - Git: 300 MB. vs Mercurial 1.2 GB for jdk/jdk repository
- Plenty of tools
 - Most IDEs have GIT integration
 - IntelliJ IDEA, Visual Studio Code, Vim. etc
- Available hosting
 - Github, Bitbucket, Gitlab etc.

Github is the most popular hosting service for GIT

- Well known in community
- Integration with external systems

Migration principles

- Keep the same history as in hg
- Migrate only single-repo projects
- Migrate JDK tools to work with Git
- Translate commit messages

JEP 394: Pattern Matching for instanceof (Standard¹⁶)

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 (Standard¹⁶)

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 360: Sealed Classes (Second Preview¹⁶)

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**

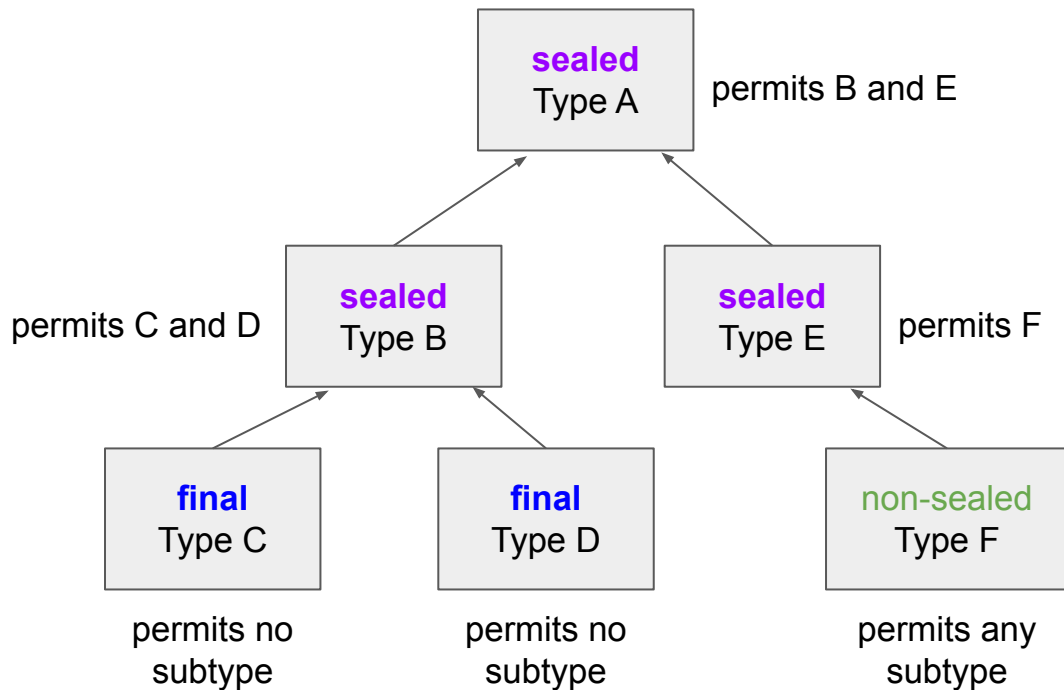
A **non-final** class

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  
    }  
}
```

JEP 389: Foreign Linker API (Incubator)

Foreign Linker API

An API to access native code in statically-typed, easy, and performant way. Can be considered as alternative to JNI (Java Native Interface), JNA (Java Native Access) etc.

```
--add-modules jdk.incubator.foreign
```

OR

```
module modulename {  
    requires jdk.incubator.foreign;  
}
```

```
-Dforeign.restricted={deny,permit,debug,warn}
```

Foreign Linker API

An API to access native code in statically-typed, easy, and performant way. Can be considered to replace JNI (Java Native Interface)

```
MethodHandle strlen = CLinker.getInstance().downcallHandle(  
    LibraryLookup.ofDefault().lookup("strlen").get(),  
    MethodType.methodType(long.class, MemoryAddress.class),  
    FunctionDescriptor.of(C_LONG, C_POINTER)  
);
```

```
MemoryAddress hello = CLinker.toCString("Hello World!").address();  
Object text = strlen.invoke(hello);  
System.out.println("Length: " + text);
```

See [Standard C Library Functions](#)

JEP 380: Unix-Domain Socket Channels

A new Socket Channel which supports unix sockets, to make inter-process communication in same machine.

Unix sockets are just internal file paths which bring less latency than the tcp/ip.

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

```
var address = UnixDomainSocketAddress.of("/usta/socketFile");  
var serverChannel = ServerSocketChannel.open(UNIX);  
serverChannel.bind(address);
```

JEP 338: Vector API (Incubator)

“Provide an initial iteration of an [incubator module](#), `jdk.incubator.vector`, to express vector computations that reliably compile at runtime to optimal vector hardware instructions on supported CPU architectures and thus achieve superior performance to equivalent scalar computations.”

(See <https://openjdk.java.net/jeps/338>)

```
--add-modules jdk.incubator.vector
```

OR

```
module modulename {  
    requires jdk.incubator.vector;  
}
```


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>

Try Java 16

Open-source builds

- <https://jdk.java.net/16>

Online Java Shell

- <https://tryjshell.org/>

Code samples

- <https://github.com/rahmanusta/java16-edu>

Thank you!