Java 14

CONTENTS

SIMON RITTER
DEPUTY CTO, AZUL

The release of JDK 14 resumes the six-month release cadence of OpenJDK and signifies that this is working very well. Although some releases have been a little light on new features (JDK 13, for example), JDK 14 includes multiple exciting new features for developers.

In addition to the usual API changes and low-level JVM enhancements, JDK 14 also includes two language preview features in the form of records and pattern matching for `instanceof`. All Java developers are familiar with `NullPointerExceptions`, and now, thankfully, they have been improved to simplify debugging.

Java has always protected developers from the kind of common errors that occur in languages like C and C++ through the use of explicit pointers that can be manipulated and point to the wrong place. Sometimes, this can be useful, and developers have resorted to using undocumented and unsupported APIs like `sun.misc.Unsafe`. In JDK 14, there is a new incubator module providing a foreign-memory access API.

RECORDS

Java is an object-oriented language; you create classes to hold data and use encapsulation to control how that data is accessed and modified. The use of objects makes manipulating complex data types simple and straightforward.

The downside (until now) is that creating a data type has been verbose, requiring a lot of code even for the most straightforward cases. Let's look at the code needed for a basic two-dimensional point *(see next column)*.

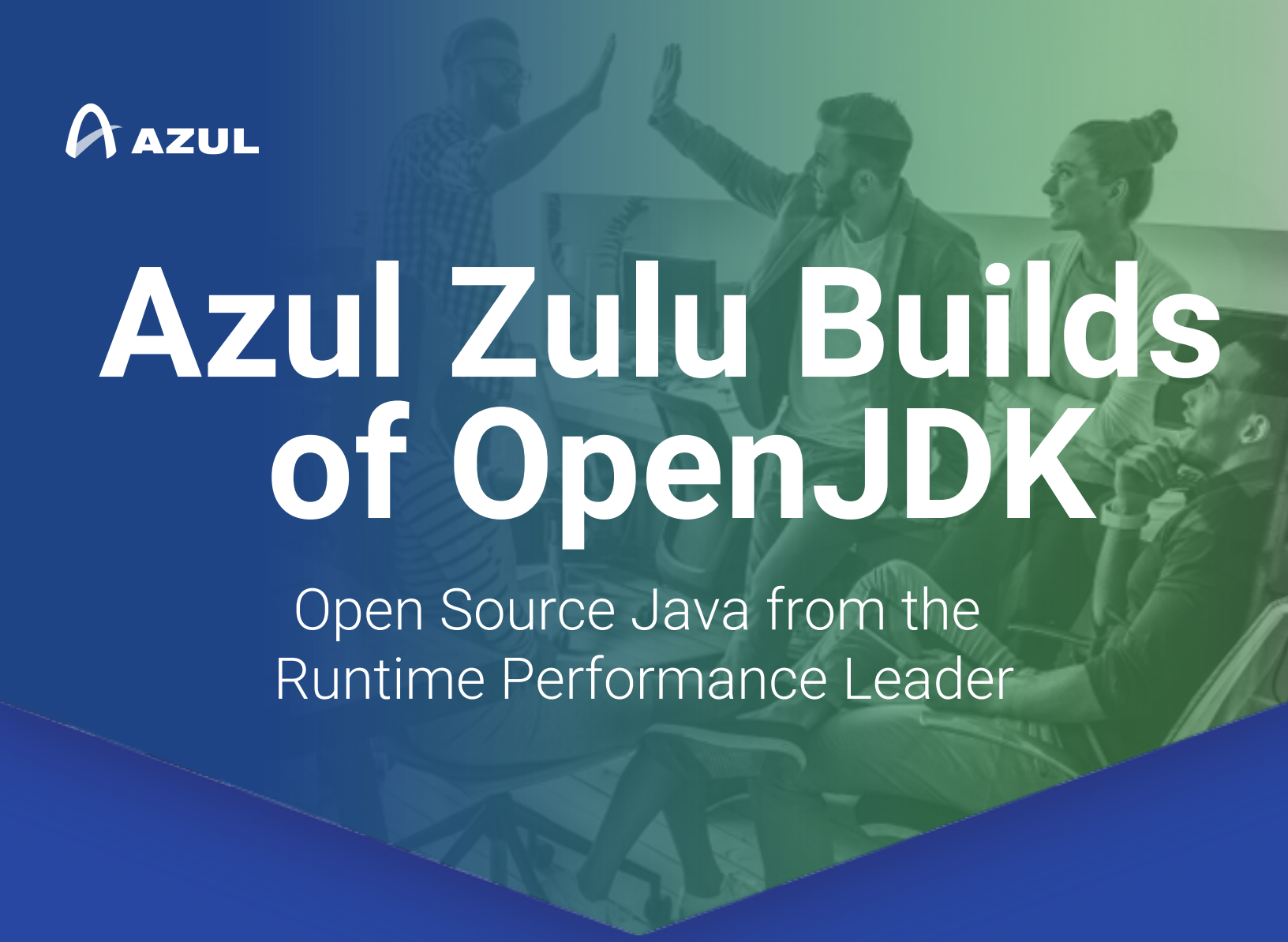
```
public class Point {
  private final double x;
  private final double y;

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

  public double getX() {
    return x;
  }

  public double getY() {
    return y;
  }
```

That's 14 lines of code just to represent a two-value tuple.



AZUL

Azul Zulu Builds of OpenJDK

Open Source Java from the Runtime Performance Leader

FREE TO DOWNLOAD AND USE WITHOUT RESTRICTIONS

COMPLIANT WITH THE JAVA SE 14,13,11,8,7,6 SPECIFICATIONS

WIDE RANGE OF SUPPORTED FILE TYPES

CHOOSE 32 OR 64-BIT JDKS AND JRES

FREE DOWNLOAD

INCLUDES:

OPEN JFX **FLIGHT RECORDER** **ZULU MISSION CONTROL**

JDK 14 introduces records as a preview feature. Preview features are a relatively new concept that allows Java platform developers to include a new language or virtual machine feature without making it part of the Java SE standard.

By doing this, developers can try out these features and provide feedback, allowing changes, if required, before the feature becomes set in the standard. To use a preview feature, you must specify the command-line flag, `--enable-preview`, for both compilation and runtime. For compilation, you must also specify the `-source` flag.

A record is a much simpler way of representing a data class. If we take our `Point` example, the code can be reduced to a single line:

```
public record Point(double x, double y) { }
```

This takes nothing away from the readability of the code; we're immediately aware that we now have something that contains two double values called `x` and `y` that we can access using the standard access or method names of `getX()` and `getY()`.

Let's examine records in greater detail.

To start with, records are a new kind of type. Records are a restricted form of class in the same way as an enum. A record has a name and a state description, which defines the components of the record.

In the `Point` example above, the state description is the doubles, `x` and `y`. Records are designed for simplicity, so they are implicitly final and cannot extend any other class (technically, all records extend the `java.lang.Record` class, preventing them from being a sub-class of anything else).

There is no restriction on a record implementing one or more interfaces. Records may not define any additional instance variables, although they can include static variables. All component state in a record is final, so no accessor (setter) methods are provided. If you need any of that, you need to use a full-blown class.

Records do have flexibility, though.

Often, the constructor needs to provide additional behavior beyond assigning values. If this is the case, we can provide an alternative implementation of the constructor:

```
record Range(int min, int max) {
  public Range {
    if (min > max)
      throw new IllegalArgumentException(“Max must be
>= min”);
  }
}
```

Note that the compact constructor definition is abbreviated, as specifying the parameters is redundant. Any of the members that are automatically derived from the state description can also be declared, so, for example, you can provide an alternative `toString()` or `hashCode()` implementation (or even in the `Point` example, the `getX()` method).

An important thing to note here is that the compact constructor must be public and can only throw an unchecked exception; there is no way to have it throw a checked exception.

PATTERN MATCHING `instanceof`

In some situations, you do not know the exact type of an object. To handle this, Java has the `instanceof` operator that can be used to test against different types. The drawback to this is that, having determined the type of an object, you must use an explicit cast if you want to use it as that type:

```
if (o instanceof String) {
  String s = (String)o;
  System.out.println(s.length);
}
```

In JDK 14, the `instanceof` operator has been extended to allow a variable name to be specified in addition to the type (this is also a preview feature). That variable can then be used without the explicit cast:

```
if (o instanceof String s)
  System.out.println(s.length);
```

The scope of the variable is limited to where its use is logically correct, so:

```
if (o instanceof String s)
  System.out.println(s.length);
else
  // s is out of scope here
```

The scope can also apply within the conditional statement, so we can do something like this:

```
if (o instanceof String s && s.length() > 4) ...
```

This makes sense since the `length()` method will only be called if `o` is a String. The same does not work with a logical or operation:

```
if (o instanceof String s || s.length() > 4) ...
```

In this case, `s.length()` needs to be evaluated, regardless of the result of whether `o` is a String. Logically, this does not work, and so, it will result in a compilation error.

Using the logical, and not the operator, can produce some interesting scoping effects:

```
if (!(o instanceof String s && s.length() > 3)
  return;
System.out.println(s.length());  // s is in scope here
```

HELPFUL `NullPointerException`

Anyone who's written more than a few lines of Java code will have experienced a `NullPointerException` at some point. Failing to initialize an object reference (or mistakenly explicitly setting it to null) and then trying to use the reference will cause this exception to be thrown.

In simple cases, finding the cause of the problem is straightforward. If we try and run code like this:

```
public class NullTest {
  List<String> list;

  public NullTest() {
    list.add("foo");
  }
}
```

The error generated is:

```
Exception in thread "main" java.lang.
NullPointerException
      at jdk14.NullTest.<init>(NullTest.java:16)
      at jdk14.Main.main(Main.java:15)
```

Since we're referencing list on line 16, it's evident that list is the culprit and we can quickly resolve the problem. However, if we use chained references in a line like this:

```
a.b.c.d = 12;
```

When we run this, we might see an error like this:

```
Exception in thread "main" java.lang.
NullPointerException
    at Prog.main(Prog.java:5)
```

The problem is that we are unable to determine, from this alone, whether the exception is as a result of a being null, b being null, or c being null. We either need to use a debugger from our IDE or change the code to separate the references onto different lines; neither of which is ideal. In JDK 14, if we run the same code, we will see something like this:

```
Exception in thread "main" java.lang.
NullPointerException:
        Cannot read field "c" because "a.b" is null
    at Prog.main(Prog.java:5)
```

Immediately, we can see that `a.b` is the problem and set about correcting it. I'm sure that this will make many Java developers lives much easier. This feature is turned off by default and requires the command-line flag `-XX:+ShowCodeDetailsInExceptionMessages` to enable it.

NEW APIs

There are 69 new API elements in the core class libraries of JDK 14. Here are the highlights:

`java.io`

There is a new annotation type, `Serial`. This is intended to be used for compiler checks on Serialization. Specifically, annotations of this type should be applied to serialization-related methods and fields in classes declared to be Serializable. (It is similar in some ways to the Override annotation.)

`java.lang`

The `Class` class has two methods for the new Record feature, `isRecord()` and `getRecordComponents()`.

The `getRecordComponents()` method returns an array of `RecordComponent` objects. `RecordComponent` is a new class in the `java.lang.reflect` package with eleven methods for retrieving things, such as the details of annotations and the generic type of each component in the record. Record is a new class that is the implicit supertype of all records and overrides the `equals`, `hashCode`, and `toString` methods of `Object`.`NullPointerException` now overrides the `getMessage` method from Throwable as part of the helpful `NullPointerExceptions` feature.

The `StrictMath` class has six new methods that supplement the existing exact methods used when overflow errors need to be detected. The new methods are `decrementExact`, `incrementExact`, and `negateExact` (all with two overloaded versions for `int` and `long`).

`java.lang.annotation`

The `ElementType` enumeration has a new constant for Records, `RECORD_TYPE`.

`java.lang.runtime`

This is a new package in JDK 14 that has a single class, `ObjectMethods`. This is a low-level part of the records feature having a single method, `bootstrap`, which generates the `equals`, `hashCode`, and `toString` methods.

`java.util.text`

The `CompactNumberFormat` class has a new constructor that adds an argument for `pluralRules`, which designates rules for associating a count keyword, such as "one", and the integer number. Its form is defined by the Unicode Consortium's Plural rules syntax.

java.util

The `HashSet` class has one new method, `toArray`, which returns an array, whose runtime component type is `Object`, containing all of the elements in this collection.

java.util.concurrent.locks

The `LockSupport` class has one new method, `setCurrentBlocker`. `LockSupport` provides the ability to park and unpark a thread (which does not suffer from the same problems as the deprecated `Thread.suspend` and `Thread.resume` methods).

It is now possible to set the object that will be returned by `getBlocker`. This can be useful when calling the no-argument park method from a non-public object.

javax.lang.model.element

The `ElementKind` enumeration has three new constants for the records and pattern matching for `instanceOf` features, namely `BINDING_VARIABLE`, `RECORD`, and `RECORD_COMPONENT`.

JEP 370: FOREIGN MEMORY ACCESS API

This is being introduced as an incubator module to allow testing by the broader Java community and feedback to be integrated before it becomes part of the Java SE standard.

It is intended as a valid alternative to both `sun.misc.Unsafe` and `java.io.MappedByteBuffer`.

The foreign-memory access API introduces three main abstractions:

- `MemorySegment`: This provides access to a contiguous memory region with given bounds.
- `MemoryAddress`: This provides an offset into a `MemorySegment` (basically, a pointer).
- `MemoryLayout`: This provides a way to describe the layout of a memory segment that greatly simplifies accessing a `MemorySegment` with a `var` handle.

  It is not necessary to calculate the offset based on the way the memory is being used. For example, an array of `int` or `long` types will offset differently but will be handled transparently using a `MemoryLayout`.

JVM CHANGES

In JDK 14, there are no functional changes to the JVM, but it does include JEPs that affect non-functional parts of the JVM.

- **JEP 345: NUMA-Aware Memory Allocation for G1** — This improves performance on large machines that use non-uniform memory architecture (NUMA).
- **JEP 363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector** — Since JDK 9, G1 has been the default collector and is considered by most to be a superior collector to the CMS.

  Given the resources required to maintain two similar profile collectors, Oracle decided to deprecate CMS (also in JDK 9), and now, it has been deleted.
- **JEP 349: JFR Event Streaming** — This allows more real-time monitoring of a JVM by enabling tools to subscribe asynchronously to Java Flight Recorder events.
- **JEP 364: ZGC on macOS and JEP 365: ZGC on Windows** — ZGC is an experimental low-latency collector that was initially only supported on Linux.

  This has now been extended to the macOS and Windows operating systems.
- **JEP 366: Deprecate the `ParallelScavenge` and `SerialOld` GC combination** — Oracle states that very few people use this combination and the maintenance overhead is considerable.

  Expect this combination to be removed at some point in the not-too-distant future.

OTHER FEATURES

There are a number of JEPs that relate to different parts of the OpenJDK:

- **JEP 343: Packaging Tool** — This is a simple packaging tool, based on the JavaFX `javapackager` tool that was removed from the Oracle JDK in JDK 11.

  This is also an incubator feature.
- **JEP 352: Non-volatile Mapped Byte Buffers** — This adds new JDK-specific file mapping mode so that the `FileChannel` API can be used to create `MappedByteBuffer` instances that refer to non-volatile memory.

  A new module, `jdk.nio.mapmode`, has been added to allow `MapMode` to be set to `READ_ONLY_SYNC` or `WRITE_ONLY_SYNC`.
- **JEP 361: Switch Expressions Standard** — Switch expressions were the first preview feature added to OpenJDK in JDK 12. In JDK 13, feedback resulted in the break value syntax being changed to yield value.

  In JDK 14, switch expressions are no longer a preview feature and have been included in the Java SE standard.

- **JEP 362: Deprecate the Solaris and SPARC ports** — Since Oracle is no longer developing either the Solaris operating system or SPARC chip architecture, they do not want to have to continue the maintenance of these ports.

  This might be picked up by others in the Java community.

- **JEP 367: Remove the Pack 200 Tools and API** — Another feature that was deprecated in JDK 11 and now removed from JDK 14. The primary use of this compression format was for jar files used by applets.

  Given the browser plugin was removed from Oracle JDK 11, this seems a reasonable idea.

- **JEP 368: Text Blocks** — These were included in JDK 13 as a preview feature, and they continue with a second iteration (still in preview) in JDK 14. Text blocks provide support for multi-line string literals. The changes are the addition of two new escape sequences.

  The first suppresses the inclusion of a newline character by putting a `\` at the end of the line (this is common in many other places in software development).

  The second is `\s`, which represents a single space. This can be useful to prevent the stripping of whitespace from the end of a line within a text block.

As you can see, JDK 14 has packed in a lot of new features that will make the lives of developers much easier.

ADDITIONAL RESOURCES

- JDK 14 Feature List
- JEP 359: Records
- JEP 305: Pattern Matching for instanceof
- JEP 370: Foreign-Memory Access API
- JDK 14 API Documentation

Written by Simon Ritter,
Deputy CTO, Azul

Simon Ritter is the Deputy CTO of Azul. Simon has been in the IT business since 1984 and holds a Bachelor of Science degree in Physics from Brunel University in the U.K.

Simon joined Sun Microsystems in 1996 and started working with Java technology from JDK 1.0; he has spent time working in both Java development and consultancy. Having moved to Oracle as part of the Sun acquisition, he managed the Java Evangelism team for the core Java platform. Now at Azul, he continues to help people understand Java as well as Azul's JVM technologies and products. Simon has twice been awarded Java Rockstar status at JavaOne and is a Java Champion. He currently represents Azul on the JCP Executive Committee and on the Java SE Expert Group.

Follow him on Twitter @speakjava.

DZone
A DEVADA MEDIA PROPERTY

DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of tousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 919.678.0300