Java 13

September 2019 saw the release of JDK 13, per Java's new release cadence of six months. In this article, we'll take a look at the new features and improvements introduced in this version.

1.. Preview Developer Features

Java 13 has brought in two new language features, albeit in the preview mode. This implies that these features are fully implemented for developers to evaluate, yet are not production-ready. Also, they can either be removed or made permanent in future releases based on feedback. We need to specify –enable-preview as a command-line flag to use the preview features. Let's look at them in-depth.

1.1 Switch Expressions (JEP 354)

We initially saw switch expressions in JDK 12. Java 13's switch expressions build on the previous version by adding a new yield statement. Using yield, we can now effectively return values from a switch expression:

```
@Test
@SuppressWarnings("preview")
public void whenSwitchingOnOperationSquareMe_thenWillReturnSquare() {
    var me = 4;
    var operation = "squareMe";
    var result = switch (operation) {
        case "doubleMe" -> {
            yield me * 2;
        }
        case "squareMe" -> {
            yield me * me;
        }
        default -> me;
    };

    assertEquals(16, result);
}
```

As we can see, it's now easy to implement the strategy pattern using the new switch.

1.2 Text Blocks (JEP 355)

The second preview feature is text blocks for multi-line Strings such as embedded JSON, XML, HTML, etc. Earlier, to embed JSON in our code, we would declare it as a String literal.

```
String JSON_STRING
  = "{\r\n" + "\"name\" : \"Baeldung\",\r\n" + "\"website\" : \"https://www.%s.com/\"\r\n" + "}";
```

Now let's write the same JSON using *String* text blocks:

```
String TEXT_BLOCK_JSON = """
{
    "name" : "Baeldung",
    "website" : "https://www.%s.com/"
}
""";
```

As is evident, there is no need to escape double quotes or to add a carriage return. By using text blocks, the embedded JSON is much simpler to write and easier to read and maintain. Moreover, all String functions are available:

```
@Test
public void whenTextBlocks_thenStringOperationsWorkSame() {
    assertThat(TEXT_BLOCK_JSON.contains("Baeldung")).isTrue();
    assertThat(TEXT_BLOCK_JSON.indexOf("www")).isGreaterThan(0);
    assertThat(TEXT_BLOCK_JSON.length()).isGreaterThan(0);
}
```

Also, String class now has three new methods to manipulate text blocks:

- stripIndent() – mimics the compiler to remove incidental white space
- translateEscapes() – translates escape sequences such as "\\t" to "\t"
- formatted() – works the same as String::format, but for text blocks

Let's take a quick look at a String::formatted example:

```
assertThat(TEXT_BLOCK_JSON.formatted("baeldung").contains("www.baeldung.com")).isTrue();
assertThat(String.format(JSON_STRING,"baeldung").contains("www.baeldung.com")).isTrue();
```

Since text blocks are a preview feature and can be removed in a future release, these new methods are marked for deprecation.

2.. Dynamic CDS Archives (JEP 350)

Class data sharing (CDS) has been a prominent feature of Java Hotspot VM for a while now. It allows class metadata to be shared across different JVMs to reduce startup time and memory footprint. JDK 10 extended this ability by adding application CDS (AppCDS) – to give developers the power to include application classes in the shared archive. JDK 12 further enhanced this feature to include CDS archives by default. However, the process of archiving application classes was tedious. To generate archive files, developers had to do trial runs of their applications to create a class list first, and then dump it into an archive. After that, this archive could be used to share metadata between JVMs. With dynamic archiving, JDK 13 has simplified this process. Now we can generate a shared archive at the time the application is exiting. This has eliminated the need for trial runs.

To enable applications to create a dynamic shared archive on top of the default system archive, we need to add an option -XX:ArchiveClassesAtExit and specify the archive name as argument:

```
java -XX:ArchiveClassesAtExit=<archive filename> -cp <app jar> AppName
```

We can then use the newly created archive to run the same app with *-XX:SharedArchiveFile* option:

```
java -XX:SharedArchiveFile=<archive filename> -cp <app jar> AppName
```

3.. ZGC: Uncommit Unused Memory (JEP 351)

The Z Garbage Collector was introduced in Java 11 as a low-latency garbage collection mechanism, such that GC pause times never exceeded 10 ms. However, unlike other Hotspot VM GCs such as G1 and Shenandoah, it was not equipped to return unused heap memory to the operating system. Java 13 added this capability to the ZGC. We now get a reduced memory

footprint along with performance improvement. Starting with Java 13, the ZGC now returns uncommitted memory to the operating system by default, up until the specified minimum heap size is reached. If we do not want to use this feature, we can go back to the Java 11 way by:

Using option -XX:-ZUncommit, or

Setting equal minimum (-Xms) and maximum (-Xmx) heap sizes

Additionally, ZGC now has a maximum supported heap size of 16TB. Earlier, 4TB was the limit.

4.. Reimplement the Legacy Socket API (JEP 353)

We have seen Socket (java.net.Socket and java.net.ServerSocket) APIs as an integral part of Java since its onset. However, they were never modernized in the last twenty years. Written in legacy Java and C, they were cumbersome and difficult to maintain. Java 13 bucked this trend and replaced the underlying implementation to align the API with the futuristic user-mode threads. Instead of **PlainSocketImpl**, the provider interface now points to **NioSocketImpl**. This newly coded implementation is based on the same internal infrastructure as java.nio. Again, we do have a way to go back to using PlainSocketImpl. We can start the JVM with the system property -Djdk.net.usePlainSocketImpl set as true to use the older implementation. The default is NioSocketImpl.

5.. Miscellaneous Changes

Apart from the JEPs listed above, Java 13 has given us a few more notable changes:

- java.nio – method FileSystems.newFileSystem(Path, Map<String, ?>) added
- java.time – new official Japanese era name added
- javax.crypto – support for MS Cryptography Next Generation (CNG)
- javax.security – property jdk.sasl.disabledMechanisms added to disable SASL mechanisms
- javax.xml.crypto – new String constants introduced to represent Canonical XML 1.1 URIs
- javax.xml.parsers – new methods added to instantiate DOM and SAX factories with namespaces support Unicode support upgraded to version 12.1
- Support added for Kerberos principal name canonicalization and cross-realm referrals
- Additionally, a few APIs are proposed for removal. These include the three String methods listed above, and the javax.security.cert API.

Among the removals include the rmic tool and old features from the JavaDoc tool. Pre-JDK 1.4 SocketImpl implementations are also no longer supported.