Java 13 Features

**Date 11/04/2023**

# Overview of Java 13 (Sept 17, 2019)

1. JEP 354 - Switch Expression
2. JEP 355 - Text Block

# JEP 354 - Switch Expression Enhancement

JEP 325 introduced switch expressions as a preview in Java 12. switch can be used as a statement or as an expression since then. "Expression" means that switch returns a value, such as in the following example from the JEP,

int numLetters = switch (day) {

case MONDAY, FRIDAY, SUNDAY:

break 6;

case TUESDAY:

break 7;

case THURSDAY, SATURDAY:

break 8;

case WEDNESDAY:

break 9;

};

Based on feedback from the developer community, **JDK Enhancement Proposal 354** replaces the break keyword in switch expressions with the new yield keyword. The new keyword is only recognized in the scope of a Switch Expression.

public class \_02\_Switch\_Expression\_Demo2 {

public static void main(String[] args) {

var a = 5;

var caseVal = "square";

var result = switch (caseVal) {

case "double":

yield a \* 2;

case "square":

yield a \* a;

default:

yield a;

};

System.***out***.println(result);

}

}

# JEP 355 - Text Block

To define a multiline string, we used to use escape sequences for line breaks and double quotes contained in the string. Java 13 introduces text blocks to handle multiline strings like JSON/XML/HTML etc. It is a preview feature. Text Block allows to write multiline strings easily without using \r\n.

Earlier, to embed JSON in our code, we would declare it as a String literal,

String JSON\_STRING

**= "{\r\n" + "\"name\" : \"Java\",\r\n" + "\"website\" : \"https://www.%s.com/\"\r\n" + "}"**

Now with text block the above string can be written as follows,

String TEXT\_BLOCK\_JSON = """

{

"name" : "Java",

"website" : "https://www.%s.com/"

}

""";

Text Block string have same methods as string like contains(), indexOf() and length() functions.

## Text Block Methods

**formatted(Object… args):** It’s similar to the String format() method. It’s added to support formatting with the text blocks.

**stripIndent():** Used to remove the incidental white space characters from the beginning and end of every line in the text block. This method is used by the text blocks and it preserves the relative indentation of the content.

**translateEscapes():** Returns a string whose value is this string, with escape sequences translated as if in a string literal.

Text blocks replaces the withdrawn **JEP 326**, "Raw String Literals", which was not accepted by the community.

public class \_01\_Text\_Block\_Demo1 {

public static void main(String[] args) {

var text\_json = """

{

"name" : "John",

"email" : "john@gmail.com"

}

""";

System.***out***.println(text\_json);

var text\_html = """

<html>

<head>

<link href='/css/style.css' rel='stylesheet' />

</head>

<body>

<h1>Hello World</h1>

</body>

</html>

""";

System.***out***.println(text\_html);

var text\_sql = """

SELECT id, firstName, lastName FROM Employee

WHERE departmentId = "IT"

ORDER BY firstName

""";

System.***out***.println(text\_sql);

}

}

public class \_02\_Text\_Block\_Demo2 {

public static void main(String[] args) {

var text\_format = """

Name: %s

Phone: %d

Salary: %.2f

""".formatted("John", 1234567890, 95000.0055);

System.***out***.println(text\_format);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

var text\_strip = "<html> \n" + "\t<body>\t\t \n" + "\t\t<p>Hello</p> \t \n" + "\t</body> \n" + "</html>";

System.***out***.println(text\_strip);

System.***out***.println(text\_strip.replace(" ", "\*"));

System.***out***.println(text\_strip.stripIndent());

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

var text\_escape = "Hi\t\nHello' \" /u0022 John\r";

System.***out***.println(text\_escape);

System.***out***.println(text\_escape.translateEscapes());

}

}

# JEP353 - Reimplement the Legacy Socket API

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.

# FileSystems.newFileSystem() Method

Three new methods have been added to the FileSystems class to make it easier to use file system providers, which treats the contents of a file as a file system.

Using the FileSystems.newFileSystem(Path path, ClassLoader loader) method, you can create a pseudo-file system with contents mapped to a file (such as a ZIP or JAR file).

The method was overloaded in Java 13 with a variant, which makes it possible to pass a provider-specific file system configuration: FileSystems.newFileSystem(Path path, Map env, ClassLoader loader)

Furthermore, two variants have been added, each without the loader parameter. A class loader is only required if the so-called FileSystemProvider for the file type to be mapped is not registered in the JDK but must be loaded via the specified class loader. For standard file types like ZIP or JAR, this is not required.

**newFileSystem(Path)**

**newFileSystem(Path, Map<String, ?>)**

**newFileSystem(Path, Map<String, ?>, ClassLoader)**

# ZGC: Uncommit Unused Memory (Experimental)

ZGC is an experimental garbage collector introduced in Java 11 that promises extremely short stop-the-world pauses of 10 ms or less.

JDK Enhancement Proposal 351 extends ZGC to return unused heap memory to the operating system after a specific time.

Using -XX:ZUncommitDelay, you can specify the time in seconds, after which ZGC returns unused memory. By default, this value is 300 seconds.

The feature is enabled by default and can be disabled with -XX:-ZUncommit.

ZGC will reach production status in Java 15. In the corresponding article, I will introduce the new garbage collector in more detail.

# References

<https://www.happycoders.eu/java/java-13-features/>

<https://www.tutorialspoint.com/java13/index.htm>

<https://www.baeldung.com/java-13-new-features>

<https://www.digitalocean.com/community/tutorials/java-13-features>