Text Blocks – JEP 355:

It allows us to create multiline strings easily. The multiline string has to be written inside a pair of triple-double quotes. The string object created using text blocks has no additional properties.

It's an easier way to create multiline strings. We can't use text blocks to create a single-line string. The opening triple-double quotes must be followed by a line terminator.

Code:

```
public class TextBlockString {

 /**
 * JEP 355: Preview Feature
 */
 @SuppressWarnings("preview")
 public static void main(String[] args) {
 String textBlock = """
 Hi
 Hello
 Yes""";

 String str = "Hi\nHello\nYes";

 System.out.println("Text Block String:\n" + textBlock);
 System.out.println("Normal String Literal:\n" + str);

 System.out.println("Text Block and String Literal equals() Comparison: " +
(textBlock.equals(str)));
 System.out.println("Text Block and String Literal == Comparison: " + (textBlock
== str));

 String textBlockHTML = """
 <html>
 <head>
 <link href='/css/style.css' rel='stylesheet' />
 </head>
 <body>
```

```
<h1>Hello World</h1>
</body>
</html>""";

String textBlockJSON = """
{
"name":"Pankaj",
"website":"JournalDev"
}""";

System.out.println("HTML ----> \n" + textBlockHTML);
System.out.println("JSON ----> \n" + textBlockJSON);
}
}
```

Output:

```
Text Block String:
Hi
Hello
Yes
Normal String Literal:
Hi
Hello
Yes
Text Block and String Literal equals() Comparison: true
Text Block and String Literal == Comparison: true
HTML ---->
<html>
<head>
<link href='/css/style.css' rel='stylesheet' />
</head>
<body>
```

```
        <h1>Hello World</h1>
    </body>
    </html>
```

JSON ---->

```
{
"name":"Pankaj",
"website":"JournalDev"
}
```

New Methods in String Class for Text Blocks:

1. formatted(Object… args): it's similar to the String format() method. It's added to support formatting with the text blocks.
2. 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.
3. translateEscapes(): returns a string whose value is this string, with escape sequences translated as if in a string literal.

Code:

```
public class StringNewMethods {
/***
* New methods are to be used with Text Block Strings
* @param args
*/
@SuppressWarnings("preview")
public static void main(String[] args) {

String output = """
Name: %s
Phone: %d
Salary: $%.2f
""".formatted("Pankaj", 123456789, 2000.5555);
```

```
System.out.println(output);

String htmlTextBlock = "<html> \n" +
"\t<body>\t\t \n" +
"\t\t<p>Hello</p> \t \n" +
"\t</body> \n" +
"</html>";
System.out.println(htmlTextBlock.replace(" ", "*"));
System.out.println(htmlTextBlock.stripIndent().replace(" ", "*"));

String str1 = "Hi\t\nHello' \" /u0022 Pankaj\r";
System.out.println(str1);
System.out.println(str1.translateEscapes());
 }
}
```

Output:

```
Name: Pankaj
Phone: 123456789
Salary: $2000.56


<html>***
<body>                  *
<p>Hello</p>**          *
</body>*
</html>
<html>
<body>
<p>Hello</p>
</body>
</html>
Hi
```

Hello' " /u0022 Pankaj

Hi

Hello' " /u0022 Pankaj

Switch Expressions Enhancements – JEP 354

Switch expressions were added as a preview feature in Java 12 release. It's almost same in Java 13 except that the "break" has been replaced with "yield" to return a value from the case statement.

Code:

```
public class SwitchEnhancements {

 @SuppressWarnings("preview")
 public static void main(String[] args) {
 int choice = 2;

 switch (choice) {
 case 1:
 System.out.println(choice);
 break;
 case 2:
 System.out.println(choice);
 break;
 case 3:
 System.out.println(choice);
 break;
 default:
 System.out.println("integer is greater than 3");
 }

 // from java 13 onwards - multi-label case statements
 switch (choice) {
 case 1, 2, 3:
 System.out.println(choice);
 break;
```

```
default:
System.out.println("integer is greater than 3");
}

// switch expressions, use yield to return, in Java 12 it was break
int x = switch (choice) {
case 1, 2, 3:
yield choice;
default:
yield -1;
};
System.out.println("x = " + x);
}

enum Day {
SUN, MON, TUE
};

@SuppressWarnings("preview")
public String getDay(Day d) {
String day = switch (d) {
case SUN -> "Sunday";
case MON -> "Monday";
case TUE -> "Tuesday";
};
return day;
}
}
```

Output:

2

2

x = 2

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.

1. newFileSystem(Path)
2. newFileSystem(Path, Map<String, ?>)
3. newFileSystem(Path, Map<String, ?>, ClassLoader)

Code:

```
import java.net.URI;
// Importing required file classes from java.nio package
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
// Importing Map and HashMap classes from java.util package
import java.util.HashMap;
import java.util.Map;

public class FileSys {

 public static void main(String[] args) {
 try {
 Map<String, String> env = new HashMap<>();
 Path testPath = Paths.get("test.txt");

 URI Uri = new URI("jar:file", testPath.toUri().getPath(), null);
 FileSystem filesystem = FileSystems.newFileSystem(Uri, env);
 System.out.println("FileSystem created successfully");

 if (filesystem.isOpen())
 System.out.println("File system is open");
 else
 System.out.println("File system is close");
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

Output:

FileSystem created successfully

File system is open