Switch Expressions

Switch Expressions after staying a preview feature in the last two releases --Java 12 and Java 13 have finally attained permanent status in Java 14.

- Java 12 introduced the lambda syntax for switch expressions thereby allowing multiple case labels for pattern matching as well as preventing fall-throughs which lead to verbose code. It also enforced exhaustive cases wherein a compilation error would be thrown if all the input cases aren't covered.
- Java 13, the second preview introduced yield statements instead of break for returning values from an expression.

Code:

```
public class SwitchExpr {
 public static void main(String[] args) {
 String day = "M";
 String result = switch (day) {
 case "M", "W", "F" -> "MWF";
 case "T", "TH", "S" -> "TTS";
 default -> {
 if (day.isEmpty())
 yield "Please insert a valid day.";
 else
 yield "Looks like a Sunday.";
 }
 };
 System.out.println(result);
 }
}
```

Output:

```
MWF
```

Pattern Matching for Instanceof

An instanceof conditional check is generally followed by a typecasting.Java 14, gets rid of this verbosity by making conditional extraction a lot more concise.
Code:

```
public class InstanceOfDemo {
 public static void main(String[] args) {
 Object obj = (Object) "abc";

 if (obj instanceof String) {
 String abc = (String) obj;
 System.out.println("Old way " + abc);
 }

 if (obj instanceof String abc) {
 //String abc = (String)obj;
 System.out.println("Is string empty " + abc.isEmpty());
 System.out.println("New way " + abc);
 }
 }
}
```

Output:

```
Old way abc

Is string empty false

New way abc
```

Helpful NullPointerException

Developers had to fall onto other debugging tools or manually figure the variable/method that was null since the stack trace would only show the line number. Java 14 introduced a new JVM feature which gives better insights with a more descriptive stack.
Code:

```
public class NPEDemo {
 public static void main(String[] args) {
 NPEDemo npObj = new NPEDemo();
 npObj.getUserName().getPass();
 }

 public NPEDemo getUserName() {
 return null;
 }

 public String getPass() {
 return "abd";
 }
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"NPEDemo.getPass()" because the return value of "NPEDemo.getUserName()" is null

at NPEDemo.main(NPEDemo.java:4)
```

TextBlockPreview

Text Blocks were introduced as a preview feature in Java 13 with the goal to allow easy creation of multiline string literals. It’s useful in easily creating HTML and JSON or SQL query strings.

- In Java 14, Text Blocks are still in preview with some new additions. We can now use Backslash for displaying nice-looking multiline string blocks.
- \s is used to consider trailing spaces which are by default ignored by the compiler. It preserves all the spaces present before it.

Code:

```
public class TextBlockPreview {
 public static void main(String[] args) {
 String text = """
 Did you know \
 Java 14
has the most features among\
 all non-LTS versions so far\
 """;

 String text2 = """
 line1
 line2 \s
 line3
 """;

 String text3 = "line1\nline2 \nline3\n";

 System.out.println("Text " + text);
 System.out.println("Text2 is equal to Text3 " + text2.equals(text3));
 }
}
```

Output:

```
Text Did you know Java 14

has the most features amongall non-LTS versions so far

Text2 is equal to Text3 false
```

Records

A record is a data class that stores pure data. The idea behind introducing records is to quickly create simple and concise classes devoid of boilerplate code.

Normally a class in Java would require you to implement equals(), hashCode() , the getters and setters methods. While some IDEs support auto-generation of such classes, the code is still verbose. With a record you need to simply define a class in the following way.

The Java compiler will generate a constructor, private final fields, accessors, equals/hashCode and toString methods automatically. The auto-generated getter methods of the above class are name() and topic().

Code:

```
import java.io.Serializable;

public record PersonRecord(String name, int age) implements Serializable {

 static int salary;

 public PersonRecord {
 if (age > 101) {
 throw new IllegalArgumentException("Unexpected ID");
 }
 }

 public static int getSalary() {
 return salary;
 }

 public static void main(String[] args) {
 var p1 = new PersonRecord("ABD", 37);
 var p2 = new PersonRecord("DEK", 32);

 System.out.println(p1.equals(p2));
 System.out.println("Sal " + getSalary());
 }
}
```

Output:

```
false

Sal 0
```