Java 14

***Switch Expressions*** *- Now be used in production code, and not just in the preview mode to be experimented with by developers. With switch expressions, we can write the same thing more succinctly:*

**boolean** isTodayHoliday = **switch** (day) {

**case** "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" -> false;

**case** "SATURDAY", "SUNDAY" -> true;

**default** -> **throw** **new** **IllegalArgumentException**("What's a " + day);

};

***Text Blocks****- text blocks now have two new escape sequences:*

*\: to indicate the end of the line, so that a new line character is not introduced*

*\s: is used to consider trailing spaces which are by default ignored by the compiler. It preserves all the spaces present before it.*

*For example*:

**String** multiline = "A quick brown fox jumps over a lazy dog; the lazy dog howls loudly."

*can now be written as*:

**String** multiline = """

A quick brown fox jumps over a lazy dog; \

the lazy dog howls loudly.""";

***Pattern Matching for instanceof -*** *Before this feature, we wrote*:

**if** (obj **instanceof** String) {

**String** str = (String) obj;

**int** len = str.length();

// ...

}

*Now, we don’t need as much code to start using*obj*as String*:

**if** (obj **instanceof** String str) {

**int** len = str.length();

// ...

}

***Helpful NullPointerExceptions -*** *Null Pointer Exceptions are a nightmare for any developer. Previously, until Java 13, it was tricky to debug the infamous NPEs. 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 as shown below*

*Exception in thread "main" java.lang.NullPointerException: Cannot invoke "Blog.getAuthor()" because the return value of "Journaldev.getBlog()" is null*

*at NullPointerExample.main(NullPointerExample.java:4)*

***Records (Preview) -*** *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. With a record you need to simply define a class in the following way.*

record Author(){}

//or

record Author (String name, String topic) {}

*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().*

*To look into the generated code, use javap <class-name> after you’ve compiled the program using javac.*

*Furthermore, we can add additional fields, methods, and constructor to the record in the following way.*

record Author (int id, String name, String topic) {

static int followers;

public static String followerCount() {

return "Followers are "+ followers;

}

public String description(){

return "Author "+ name + " writes on "+ topic;

}

public Author{

if (id < 0) {

throw new IllegalArgumentException( "id must be greater than 0.");

}

}

}

*The additional constructor defined inside the record is called a Compact constructor. It doesn’t consist of any parameters and is just an extension of the canonical constructor.*

*A compact constructor wouldn’t be generated as a separate constructor by the compiler. Instead, it is used for validation cases and would be invoked at the start of the main constructor.*

*Few important things to note about Records:*

* *A record can neither extend a class nor it can be extended by another class. It’s a final class.*
* *Records cannot be abstract*
* *Records cannot extend any other class and cannot define instance fields inside the body. Instance fields must be defined in the state description only*
* *Declared fields are private and final*
* *The body of a record allows static fields and methods.*
* *Values Inside Reference Fields Of A Record Can Be Mutated - It’s important to note that for fields defined which are objects, only the reference is immutable. The underlying values can be modified. Ex. List declared in record can be modified, reference will be final.*
* *Records Can Implement Interfaces.*
* *Records support multiple constructors.*
* *Records Allow Modifying Accessor Methods - Though records do generate public accessor methods for the fields defined in the state description, they also allow you to redefine the accessor methods. i.e. Getter methods are generated, but those can be modified.*

*Records are designed as plain data carriers and if you’re looking to implement a lot of additional methods, it’s better to fall back onto the normal class.*