Core Java Updates

Refer Java Language Changes on each version:

https://docs.oracle.com/en/java/javase/21/language/java-language-changes.html#GUID-EE327F97-BF08-46D2-8426-085CBE585797

Lesson-1

JDK 21 – **Unnamed Classes and Instance Main Methods**

https://docs.oracle.com/en/java/javase/21/language/unnamed-classes-and-instance-main-methods.html#GUID-E49690F1-727E-45F6-A582-9821C9597112

HelloWorld.java

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello World");
        }
}
```

Instead of traditional code, if you are working with JDK 21, you can run your code with the below signature. To say Hello World.

```
void main(){
   System.out.println("Hello World");
}
```

Lesson-2

About var [LVTI] – Local Variable Type Inference

Documentation: https://docs.oracle.com/en/java/javase/22/language/local-variable-type-inference.html

https://openjdk.org/projects/amber/guides/lvti-faq

- Starting with JDK 10, Java introduced a new way to declare local variables called local variable type inference.
- This feature allows you to use the **var** identifier to declare a variable without explicitly specifying its type.
- The type of the variable is inferred by the compiler based on the type of the initializer expression.

- This feature is only applicable to local variables within a method or any code block, including for loops, try-with-resources, and catch blocks.
- Since **var** is not a keyword, most existing code that uses **var** as a variable, method, or package name will not break. Java intentionally made this design choice to maintain backward compatibility with older versions of Java where **var** might have been used as an identifier.
- Since **var** is reserved as a type name, you cannot use **var** to name a class or interface.
- Using **var** can make Java code cleaner and more readable

```
int x = 10;
 // Local variables are declared inside a method.
 var y = 10; // Infer int automatically
 // cannot reassign other type of values cause error
// y = 12.5;
 var name = "Anonymous"; // Infer as a String automatically
 System.out.println("length = " + name.length());
 var sum=0;
 for (var counter = 0; counter < 10; counter++)
    sum+=counter;
 System.out.println("Sum = " + sum);

 var person = new Person();
```

Cannot use var in the following areas,

- Method Parameters
- Field or instance variables
- Return types for methods
- Parameters for constructors
- Lambda parameters
- Array declarations

Refer Demo: lesson2.varjdk10

RandomGenerator Interface

Oracle Documentation: https://blogs.oracle.com/javamagazine/post/java-pseudo-random-number-generator-enhancements

- Introduced in JDK 17
- Part of the java.util.random package.

- Designed to offer a uniform API for all random number generators.
- Supports multiple types of random number generators.
- Includes methods for generating streams of random numbers. [Streams will be discussed in MPP]
- Offers additional methods for producing random numbers within a specific range.
- Facilitates easy switching between different random number generation algorithms.
- Ensures higher security in applications needing robust randomness, e.g., SecureRandom for cryptographic operations.

  RandomGenerator secureRandom = SecureRandom.getInstanceStrong();
  System.out.println(secureRandom.nextExponential());

Refer Demo: lesson2.random.RandomJDK17.java

Text Block [JDK 15]

A text block's is to provide clarity by way of minimizing the Java syntax required to render a string that spans multiple lines.

```
String colors = """
   red
   green
   blue
   """;
```

is equivalent to "red\ngreen\nblue\n".

Refer Demo: lesson2.jdkUpdatesString

String Template[JDK 21]

Documentation: https://docs.oracle.com/en/java/javase/22/language/string-templates.html#GUID-CAEF15BD-C3D1-43D4-B38F-1615B0B1699D

- String Template is an Interface and introduced in JDK 21
- String templates complement Java's existing string literals and text blocks by coupling literal text with embedded expressions and template processors to produce specialized results.
- An *embedded expression* is a Java expression except it has additional syntax to differentiate it from the literal text in the string template.
- A *template processor* combines the literal text in the template with the values of the embedded expressions to produce a result.

```
var x = 10;
var y = 20;
// If I want to print 10 + 20 = 30
```

```
System.out.println(x + " + " + y + " = " + (x + y));
// Simplified with String Template, STR is a //
template processor
System.out.println(STR."\{x} + \{y} = \{x + y}");
```

Refer Demo: lesson2.jdkUpdatesString

Switch Expression

Documentation: https://docs.oracle.com/en/java/javase/21/language/switch-expressions.html

- switch expressions introduced in Java 12 as a preview feature and standardized in Java 14.
- Eliminate the need for break statements to prevent fall through.
- Can be used as an expression that returns a value, simplifying code.
- In the enhanced switch, use -> instead of : in the respective case.
- You can use a yield statement to specify the value of a switch expression.
- Allows executing multiple statements(block) before yielding a result.
- default is provided to handle unexpected values.
- You cannot use : and -> together. You can use any one in all cases.
- In Java 21, you have a convenient way to handle the null case explicitly by writing, case null -> – which lets you avoid the NPE(NullPointerException).

Example:

```
var w = 145.6;
double weight=0.0;
String p = "venus";
p = p.toLowerCase();

switch (p) {
    case "venus" -> weight = w * 0.78;
    case "mars" -> weight = w * 0.39;
    case "jupiter" -> weight = w * 2.65;
    case "saturn" -> weight = w * 1.17;
    case "uranus" -> weight = w * 1.05;
    case "neptune" -> weight = w * 1.23;
    default -> System.out.println("Wrong choice");
}
```

Refer Demo: lesson2.switchdemo.SwitchExpression

Lesson-5

JDK 14 - Pattern matching for Instance of

It simplify the way developers handle type checking and casting in Java. This feature reduces boilerplate code and increases the readability and safety of Java code.

Pattern matching introduces new language enhancements that enable you to conditionally extract data from objects with code that's more concise and robust.

Traditional Approach

```
if (obj instanceof String) { // Checking the type
   String s = (String) obj; // Casting
   // use s
}
```

JDK-14 instanceof

```
if (obj instanceof String s) {
   // use s directly here
}
```

- Here, s is automatically cast to String if obj is an instance of String.
- Eliminate the need of writing a code line for casting.
- If obj is not an instanceof String, body will not be executed. Here, s is called pattern variable and a local variable.

Refer Demo: lesson5\instanceoff

Lesson 11 - Creating Unmodifiable Collection

https://docs.oracle.com/en/java/javase/17/core/creating-immutable-lists-sets-and-maps.html#GUID-1222F8A3-7EC0-4E49-9B75-C3B263F9A1BB