Java 17

Links :-

1. http://openjdk.java.net/projects/jdk/17/
2. https://cr.openjdk.java.net/~iris/se/17/latestSpec/apidiffs/overview-summary.html (API differences)
3. https://www.oracle.com/java/technologies/javase/17-relnote-issues.html#NewFeature

Random Generator:-

1. Provided new interfaces for pseudo random number generator(PRNG).
2. There are stream of implementations that we can use to generate random numbers.
   - L32X64MixRandom
   - L32X64StarStarRandom
   - L64X128MixRandom
   - L64X128StarStarRandom
   - L64X256MixRandom
   - L64X1024MixRandom
   - L128X128MixRandom
   - L128X256MixRandom
   - L128X1024MixRandom
3. For Ex:-

RandomGeneratorExample.java

```java
package java17.rndm;

import java.util.random.RandomGenerator;
import java.util.random.RandomGeneratorFactory;

public class RandomGeneratorExample {

    public static void main(String[] args) {
        RandomGeneratorFactory.all()
                .toList().forEach(r -> System.out.println(r.name() + " :- " + r.group()));

        RandomGenerator rndmGenrator = RandomGeneratorFactory.of("L64X1024MixRandom").create();
        for(int i = 0; i < 10; i++)
            System.out.println(rndmGenrator.nextInt( bound: 20));
    }
}
```

Run: RandomGeneratorExample

```
D:\Mukesh\Softwares\Java\Java-17\jdk-17.
L32X64MixRandom :- LXM
L128X128MixRandom :- LXM
L64X128MixRandom :- LXM
SecureRandom :- Legacy
L128X1024MixRandom :- LXM
L64X128StarStarRandom :- LXM
Xoshiro256PlusPlus :- Xoshiro
L64X256MixRandom :- LXM
Random :- Legacy
Xoroshiro128PlusPlus :- Xoroshiro
L128X256MixRandom :- LXM
SplittableRandom :- Legacy
L64X1024MixRandom :- LXM
10
10
12
11
15
1
15
11
19
10
```

4. We can use any of the listed random generator factory.

Switch Expression Enhancement:-

1. The biggest change of Java-17, we can pass super class object in switch expression and case statements take care of all sub classes. Thanks to the pattern matching and sealed class concept. For Ex:-

```
return seitch(SuperClass) {
    case Chield1 c1 -> some operation;
    case Chield2 c2 && c2.method() > some_val -> some operation;
    case Chield3 c3 -> some operation;
    default: throw new IllegalArgumentException("Invalidate class is passed.");
};
```

2. Lets have an example for better understanding:-

```
public static void main(String[] args) {
    System.out.println("Area :- " + getArea(new Circle( radius: 10)));

    System.out.println("Perimeter :- " + getPerimeter(new Rectangle( length: 10,  width: 15.78)));
}

private static double getPerimeter(GeometricalCalculation geoCal) {
    return switch (geoCal) {
        case Circle c -> c.perimeter();
        case Square s -> s.perimeter();
        case Rectangle r -> r.perimeter();
        default -> throw new IllegalStateException("Unexpected value: " + geoCal);
    };
}

private static double getArea(GeometricalCalculation geoCal) {
    if(geoCal instanceof Circle c) {
        return c.area();
    } else if(geoCal instanceof Square s) {
        return s.area();
    } else if(geoCal instanceof Rectangle r) {
        return r.area();
    }
    throw new IllegalArgumentException("Invalid argument passed:- " + geoCal);
}
```

```
D:\Mukesh\Softwares\Java\Java-17\jdk-17.
Area :- 314.1592653589793
Perimeter :- 51.56

Process finished with exit code 0
```

3. In above example, we can see that lots of boilerplate code is removed.
4. Chance of IllegalStateException is ZERO.
5. Code readability is increased.

6. We can also add conditions within the case statements. For example:-

```java
public static void main(String[] args) {
    System.out.println("Area :- " + getArea(new Circle( radius: 10)));

    System.out.println("Perimeter :- " + getPerimeter(new Rectangle( length: 10,  width: 15.78)));
}

private static double getPerimeter(GeometricalCalculation geoCal) {
    return switch (geoCal) {
        case Circle c && c.radius() > 5 -> c.perimeter();
        case Square s -> s.perimeter();
        case Rectangle r -> r.perimeter();
        default -> throw new IllegalStateException("Unexpected value: " + geoCal);
    };
}

private static double getArea(GeometricalCalculation geoCal) {
    if(geoCal instanceof Circle c) {
        return c.area();
    } else if(geoCal instanceof Square s) {
        return s.area();
    } else if(geoCal instanceof Rectangle r) {
        return r.area();
    }
    throw new IllegalStateException("Invalid argument passed:- " + geoCal);
}
```

```
D:\Mukesh\Softwares\Java\Java-17\jdk-17.
Area :- 314.1592653589793
Perimeter :- 51.56

Process finished with exit code 0
```

7. As per case statement of Circle, we have added a condition here. It can be more complex as per need.
8. We have to add all sub class as a case or a default case(With at least one case) otherwise it is a compilation error. After sealed class/interface concept, all child classes are know to java compiler.
9. Null is also supported as an case from java-17. Earlier we got NullPointerException.

```java
public static void main(String[] args) {
    System.out.println("Area :- " + getArea(new Circle( radius: 10)));
    System.out.println("Perimeter :- " + getPerimeter(new Rectangle( length: 10,  width: 15.78)));
    System.out.println("Null:- " + getPerimeter( geoCal: null));
}
private static double getPerimeter(GeometricalCalculation geoCal) {
    return switch (geoCal) {
        case Circle c && c.radius() > 5 -> c.perimeter();
        case Square s -> s.perimeter();
        case Rectangle r -> r.perimeter();
        case null -> 0;
        default -> throw new IllegalStateException("Unexpected value: " + geoCal);
    };
}
private static double getArea(GeometricalCalculation geoCal) {
    if(geoCal instanceof Circle c) {
        return c.area();
    } else if(geoCal instanceof Square s) {
        return s.area();
    } else if(geoCal instanceof Rectangle r) {
        return r.area();
    }
    throw new IllegalStateException("Invalid argument passed:- " + geoCal);
}
```

```
D:\Mukesh\Softwares\Java\Java-17\jdk-17.
Area :- 314.1592653589793
Perimeter :- 51.56
Null:- 0.0

Process finished with exit code 0
```

10.