

Java 12

Links :-

1. <http://openjdk.java.net/projects/jdk/12/>
2. <https://www.azul.com/39-new-features-and-apis-in-jdk-12/>
3. <http://cr.openjdk.java.net/~iris/se/12/latestSpec/apidiffs/overview-summary.html>
(API differences)
4. <https://www.oracle.com/java/technologies/javase/12-relnote-issues.html#NewFeature>
- 5.

Language Changes :-

1. **String class changes :-**
 - `indent(positiveInteger)` :- Add spaces in front of the string as provided in argument.
 - `indent(negativeNumber)`:- Remove spaces from the front of string. If spaces are lesser than the provided number then it will ignore rest of them.
 - `transform(Function)`:- This method will apply the supplied function on the string.
2. **Switch Expression (<http://cr.openjdk.java.net/~gbierman/jep354-jls-201905.html>) :-** This is the biggest change in Java-12 and not permanent as of now. If it breaks the backward compatibility then it can be removed in higher versions.
 - Newer switch expression is more clear and easy to understand.
 - Now switch can be used as an statement or as an expression.
 - Means now switch can return a value as a result.
 - Following syntaxes are supported :-

- Newer version :- This syntax is used in case of single line body of case.

```
private static void newVersion(int num) {  
  
    switch (num) {  
        case 1,2,3 -> System.out.println("1 to 3 : NEW");  
        case 4 -> System.out.println("Only 4 : NEW");  
        case 5,6 -> System.out.println("5 and 6 : NEW");  
        default -> System.out.println("Default : NEW");  
    };  
}
```

- Return value via switch :-

```
private static void retrunValueViaArrow(int num) {  
    var status = switch (num) {  
        case 1,2,3 -> 10;  
        case 4 -> 20;  
        case 5,6 -> 30;  
        default -> num;  
    };  
  
    System.out.println("Status :- " + status);  
}
```

- Multiple case with break using newer format :- More than 1 lines.

```
private static void multipleCaseWithBreak(int num) {  
    var status = 0;  
  
    switch (num) {  
        case 1,2,3:  
            status = 10;  
            break;  
        case 4:  
            status = 20;  
            break;  
        case 5,6:  
            status = 30;  
            break;  
        default:  
            status = num;  
    }  
  
    System.out.println("multipleCaseWithBreak :- " + status);  
}
```

- Break statement can also be used to return a value :-

```
static int damageAgain(String plastic) {  
    return switch (plastic) {  
        case "SPOON" -> 10;  
        case "STRAW" -> {  
            int number = 100;  
            // some code that manipulates this value  
            // connect to a DB..  
            break 10;  
        }  
        default -> 100;  
    };  
}
```

- We can also define a body of case block :-

```
private static void switchWithBlock(int num) {  
    switch (num) {  
        case 1,2,3 ->{  
            var n = 1000;  
            // Do some operation.  
            // DB connection etc.  
            System.out.println("1 to 3 With block + n :- " + n);  
        }  
        case 4 -> System.out.println("Only 4 : NEW");  
        case 5,6 -> System.out.println("5 and 6 : NEW");  
        default -> System.out.println("Default : NEW");  
    }  
}
```

- Switch expression should cover all the possible values. Otherwise it is an compilation error.












3. **Constant APIs :-**

- Java-12 comes with some Constant APIs.

- These APIs are under "java.lang.constant" package.

Package Detail

(*) Columns: (1) - Changed, (2) - Added, (3) - Removed, Σ - Total

File	Contexts					Total
	Modified (*)				Un- changed	
	1	2	3	Σ		
 ClassDesc		16		16	0	16
 Constable		2		2	0	2
 ConstantDesc		2		2	0	2
 ConstantDescs		58		58	0	58
 DirectMethodHandleDesc		7		7	0	7
 DirectMethodHandleDesc.Kind		18		18	0	18
 DynamicCallSiteDesc		14		14	0	14
 DynamicConstantDesc		15		15	0	15
 MethodHandleDesc		8		8	0	8
 MethodTypeDesc		15		15	0	15
 package-summary		1		1	0	1
Total	0	156	0	156	0	156

© 2019 Oracle Corporation and/or its affiliates

- There are few that have been implemented are :-
 - Object resolveConstantDesc(MethodHandles.Lookup lookup) -> ConstantDesc
 - Optional<? extends ConstantDesc> describeConstable() -> Constable

4. JVM Changes :-

- Shenandoah GC :- <https://blogs.oracle.com/javamagazine/understanding-the-jdks-new-superfast-garbage-collectors>
 - It is an experimental Garbage Collector.
 - It is a low time-pause garbage collector.
 - It is proposed and developed via Red Hat.
 - It reduces the pause time of the GC by doing evacuation work concurrently with the running java application.
 - Pause time with Shenandoah is independent of heap size. The size of heap will never impact. Whether it is of 200 MB or 200 GB.

- G1 does same parallel and concurrent processing but it does not do concurrent evacuation.
- Modern machines have more memory and more processors than ever before. Service Level Agreement (SLA) applications guarantee response times of 10-500ms. In order to meet the lower end of that goal we need garbage collection algorithms which are efficient enough to allow programs to run in the available memory, but also optimized to never interrupt the running program for more than a handful of milliseconds. Shenandoah is an open-source low-pause time collector for OpenJDK designed to move us closer to those goals.
- Abortable Mixed Collection For G1 :-
 - Young Region :- Frequently created objects.
 - Old Region :- Objects that are in memory from a long time.
 - Mixed Region :- Mix of both Young and Old Region.
 - Whenever G1 is about to go out of memory then it first look for Young Region. If application is still facing memory issues then it will look at Mixed Region.
 - Before Java-12 it was not possible to abort mixed collection for G1. But from Java-12 and on-wards it is possible.
- Promptly returns unused committed memory from G1 :-
 - G1 only returns the memory when Full Garbage Collection is performed. It is done only in case when there is no more space to allocate the memory to new objects.
 - But from Java-12 G1 do one additional task and it will return unused memory frequently in an particular amount of time and on a particular event.
 - This will get know to host about available free memory.

5. Micro Bench Marking Suite :-

- This is useful for performance compositions.

6.

