Java 12 Features:
----
1. Switch Expressions
2. File mismatch() Method
3. Compact Number Formatting
4. Teeing Collectors in Stream API
5. Java Strings New Methods - indent(), transform(), describeConstable(), and resolveConstantDesc().
6. JVM Constants API
7. Pattern Matching for instanceof
8. Raw String Literals is Removed From JDK 12.

Note:
IN Java12 version, some features are preview features, we must compile and execute that preview features by enable preview features. To enable preview features for compilation and execution we have to use the following commands.

```
javac --enable-preview -source 12 Test.java
java --enable-preview Test
```

1. Switch Expressions:

It is preview feature only, it will be included in the future versions.

Upto Java 11 version, we are able to write switch as statement like below.

```
switch(var)
{
   case val1:
     ----instuctions----
   break;
   case val2:
      ----instructions----
   break;
   ----
   ----
   case n:
      ---instructions----
   break;
   default:
     ----instructions----
   break;
}
```

From Java12 version onweards, we are able to use switch in the following two ways.
1. Switch Statement
2. Switch Expression.

1. Switch Statement:

It is almost all same as switch statement before Java12 version, but, in JAVA12 version, we are able to define multi labelled case statements in switch.
Syntax:

```
switch(val){
  case label1, label2,...label_n:
        -------
  break;
  ------
  -------
  default:
       ------

  break;
}
```

EX:

```
class Test
{
```

```
{
        public static void main(String[] args)
        {
                var i = 10;
                switch(i){
                        case 5,10:
                                System.out.println("Five or Ten");
                        break;

                        case 15, 20:
                                System.out.println("Fifteen or Twenty");
                        break;

                        default:
                                System.out.println("Defasult");
                        break;
                }
        }
}
OP:
----
Five or Ten

2. Switch Expression:
----------------------
In JAVA12 version, we are able to use switch as an expression, it return a value
and it is possible to assign that value to any variable.

In switch expression, switch is able to return values in two ways.
1. By Using break statement.
2. By using Lambda style syntax.

1. By Using break statement:
-----------------------------
Syntax:
--------
var varName = switch(value){
    case val1:
        break value;
    case val2:
        break value;
    -----
    -----
    case val_n:
        break value;
    default:
        break value;
}; // ; is mandatory
EX:
----
EX:
----
class Test
{
        public static void main(String[] args)
        {
                var i = 10;
                var result = switch(i){
                        case 5:
                           break "Five";
                        case 10:
                           break "Ten";
                        case 15:
                           break "Fifteen";
                        case 20:
                           break "Twenty";
                        default:
                                break "Number not in 5, 10, 15 and 20";
                };
                System.out.println(result);
        }
```

```
        }
}
OP:
----
Ten
```

Note: In switch expression, we are able to provide multi lebelled cases.

EX:

```
class Test
{
        public static void main(String[] args)
        {
                var i = 10;
                var result = switch(i){
                        case 5, 10:
                                break "Five or Ten";
                        case 15,20:
                                break "Fifteen or Twenty";

                        default:
                                break "Number not in 5, 10, 15 and 20";
                };
                System.out.println(result);
        }
}
OP:
---
Five or Ten
```

2. By using Lambda style syntax:

```
-------------------------------
Syntax:
-------
DataType result = switch(value){
   case val1 -> Expression;
   ----
   default -> Expression;
};
EX:
---
class Test
{
        public static void main(String[] args)
        {
                var i = 10;
                var result = switch(i){
                        case 5 -> "Five";
                        case 10 -> "Ten";
                        case 15 -> "Fifteen";
                        case 20 -> "Twenty";
                        default -> "Number not in 5, 10, 15 and 20";
                };
                System.out.println(result);
        }
}
OP:
----
Ten
```

Note: In switch, Lambda style syntax, we are able to provide multi labelled cases.

Syntax:

```
--------
DataType result = switch(value){
   case val1,val2,...val_n -> Expression;
   ----
   default -> Expression;
};
```

EX:

```
---
class Test
{
        public static void main(String[] args)
        {
                var i = 10;
                var result = switch(i){
                        case 5, 10 -> "Five or Ten";
                        case 15,20 -> "Fifteen or Twenty";
                        default ->  "Number not in 5, 10, 15 and 20";
                };
                System.out.println(result);
        }
}
OP:
---
Five or Ten

EX:
---
import java.io.*;
class Test
{
        public static void main(String[] args)
        {
                var day = "";
                var dayType = "";
                try{
                        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
                        System.out.print("Enter Day [Upper case letters only]  :
");
                        day = br.readLine();
                        dayType = switch(day){
                                case
"MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY" -> "WEEK DAY";
                                case "SATURDAY","SUNDAY" -> "WEEKEND";
                                default -> day+" is an invalid Day";
                        };
                }catch(Exception e){
                        System.out.println(e.getMessage());
                }
                System.out.println(day+" is  "+dayType);
        }
}

To compile and execute  the above code we must enable preview feature along with
javac and java

D:\java7\java12Features>javac --enable-preview -source 12 Test.java
Note: Test.java uses preview language features.
Note: Recompile with -Xlint:preview for details.

D:\java7\java12Features>java --enable-preview Test
Enter Day [Upper case letters only]  : MONDAY
MONDAY is  WEEK DAY

D:\java7\java12Features>java --enable-preview Test
Enter Day [Upper case letters only]  : SUNDAY
SUNDAY is  WEEKEND

D:\java7\java12Features>java --enable-preview Test
Enter Day [Upper case letters only]  : MNDAY
MNDAY is an invalid Day
MNDAY is


2. Files mismatch() Method:
--------------------------
JAVA12 version has provided mismatch() method in java.nio.file.Files class, it
```

can be used to check whether two files content is matched or not, if files content is matched then mismatch() method will return -1L value , if the files content is mismatched then mismatch() method will return the position of the mismatched byte.

EX:

```
---
package java12features;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class Test {

        public static void main(String[] args)throws Exception {

                Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
                Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
                Files.writeString(file1, "Welcome To Durga Software Solutions");
                Files.writeString(file2, "Welcome To Durga Software Solutions");
                long val = Files.mismatch(file1, file2);
                System.out.println(val);
                if(val == -1L) {
                        System.out.println("Files Content is Matched ");
                }else {
                        System.out.println("Files Content isMismatched `");
                }
        }
}
```

OP:

```
---
-1
Files Content is Matched
```

EX:

```
---
package java12features;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class Test {

        public static void main(String[] args)throws Exception {

                Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
                Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
                Files.writeString(file1, "Welcome To Durga Software Solutions");
                Files.writeString(file2, "Welcome To Durgasoft");
                long val = Files.mismatch(file1, file2);
                System.out.println(val);
                if(val == -1L) {
                        System.out.println("Files Content is Matched ");
                }else {
                        System.out.println("Files Content isMismatched `");
                }
        }
}
```

OP:

```
---
16
Files Content isMismatched
```

3. Compact Number Formatting:

In General, in java applications, we are able to represent numbers in numberic form, not in Short form like 10K, 100K,..... and Long form like 1 Thousand or 100 Thousand, if we want to represent numbers in short form and long form like above JAVA12 version has provided a predefined class in the form of

above JAVA12 version has provided a predefined class in the form of
java.text.CompactNumberFormat.

By Using CompactNumberFormat we are able to perform the following actions.
1. Converting Normal Numbers to Short Form and Long Form of Numbers
   100 -------> 100
   1000 ------> 1 Thousand or 1 k
   10000 -----> 10 Thousand or 10 k

2. Converting Numbers from Short or long form to Numeric values.
   100 -------> 100
   1 k -------> 1000
   10 K ------> 10000

   100 -------> 100
   1 thousand -------> 1000
   10 thousand ------> 10000

To get CompactNumberFormat object we have to use the following Factory method
from NumberFormat class.
   CompactNumberFormat cnf = NumberFormat.getCompactNumberInstance(Locale l, int
style)
   Where Style may be NumberFormat.style.SHORT or NumberFormat.style.LONG

To convert a number to Compact Number we will use the following method.
   public String format(Number num)

To convert compact number to number we will use the following method.
   public Number parse(String compactNumber)

EX:

```
package com.durgasoft.java12features;

import java.text.NumberFormat;
import java.util.Locale;

public class Test {

        public static void main(String[] args)throws Exception {
                // Converting value from Normal number to Compact Number in
Short form
                NumberFormat shortForm =
NumberFormat.getCompactNumberInstance(new Locale("en",
"US"),NumberFormat.Style.SHORT);
                System.out.println("1000 ----> "+shortForm.format(1000));
                System.out.println("10000 ---> "+shortForm.format(10000));
                System.out.println("100000 --> "+shortForm.format(100000));
                System.out.println();

                // Converting value from Normal number to Compact Number in Long
form
                NumberFormat longForm =
NumberFormat.getCompactNumberInstance(new Locale("en",
"US"),NumberFormat.Style.LONG);
                System.out.println("1000 ----> "+longForm.format(1000));
                System.out.println("10000 ---> "+longForm.format(10000));
                System.out.println("100000 --> "+longForm.format(100000));

                // Converting value from Compact number in Long Form to Normal
Number
                System.out.println();
                NumberFormat numFormat =
NumberFormat.getCompactNumberInstance(new Locale("en",
"US"),NumberFormat.Style.LONG);
                System.out.println("1 thousand -----> "+numFormat.parse("1
thousand"));
                System.out.println("10 thousand ----> "+numFormat.parse("10
thousand"));
                System.out.println("100 thousand ---> "+numFormat.parse("100
```

```
thousand"));

                // Converting value from Compact number in Short Form to Normal
Number
                System.out.println();
                NumberFormat numFmt = NumberFormat.getCompactNumberInstance(new
Locale("en", "US"),NumberFormat.Style.SHORT);
                System.out.println("1k -----> "+numFmt.parse("1k "));
                System.out.println("10k ----> "+numFmt.parse("10k"));
                System.out.println("100k ---> "+numFmt.parse("100k"));


        }
}
```

OP:
```
1000 ----> 1K
10000 ---> 10K
100000 --> 100K

1000 ----> 1 thousand
10000 ---> 10 thousand
100000 --> 100 thousand

1 thousand -----> 1000
10 thousand ----> 10000
100 thousand ---> 100000

1k -----> 1
10k ----> 10
100k ---> 100
```

4. Teeing Collectors in Stream API

JAVA 12 version has provided Teeing collector in Stream API, its main purpose is to take two straems and performing BIFunction then generating results.

```
public static Collector teeing (Collector stream1, Collector stream2, BiFunction
merger);
```
Where stream1 and Stream2 are two Streams and merger is able to merge the rejjsult of both Collectors and generating result.

EX:
```
package java12features;

import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Test {

        public static void main(String[] args)throws Exception {
                double mean = Stream.of(1,2,3,4,5,6,7,8,9,10).collect(
                                Collectors.teeing(
                                                Collectors.summingDouble(x->x),
                                                Collectors.counting(),
                                                (sum,count)->sum/count));
                System.out.println("Mean : "+mean);


        }
}
```

OP:
```
Mean : 5.5
```

5. Java Strings New Methods

JAVA12 version has introduced the following new functions in String class.
1.indent(),
2.transform()

1.public String indent(int count)
--> The main intention of ident() method is to add spaces from a string or remove spaces to the string.

1. If count < 0 then spaces will be removed at the begining of each and every line.
2. If count > 0 then spaces will be added at beginning of String.
3. If negative count > the existed spaces then all spaces are removed.

EX:

```
package java12features;
public class Test {
        public static void main(String[] args)throws Exception {
                String str1 = "Durga\nSoftware\nSolutions";
                System.out.println(str1);
                String newString1 = str1.indent(5);
                System.out.println(newString1);
                System.out.println();
                String str2 = "     Durga\n     Software\n     Solutions";
                System.out.println(str2);
                String newString2 = str2.indent(-5);
                System.out.println(newString2);
                System.out.println();
                String str3 = "   Durga\n   Software\n   Solutions";
                System.out.println(str3);
                String newString3 = str3.indent(-5);
                System.out.println(newString3);
                System.out.println();
                String str4 = "   Durga\n   Software\n   Solutions";
                System.out.println(str4);
                String newString4 = str4.indent(0);
                System.out.println(newString4);

        }
}
```

OP:

```
Durga
Software
Solutions
     Durga
     Software
     Solutions


     Durga
     Software
     Solutions
Durga
Software
Solutions


   Durga
   Software
   Solutions
Durga
Software
Solutions


   Durga
```

```
   Software
   Solutions
   Durga
   Software
   Solutions
```

2. public R transform(Function f)
--> It will take a Function as parameter and it will transform string into some other form and return that result.
EX:

```
package java12features;
public class Test {
        public static void main(String[] args)throws Exception {
                String str = "Durga Software Solutions";
                String newString = str.transform(s->s.toUpperCase());
                System.out.println(newString);
        }
}
```
OP:

```
DURGA SOFTWARE SOLUTIONS
```

6. JVM Constants API:

JAVA12 version has introduced constants API, it includes two interfaces like java.lang.constant.Constable and java.lang.constant.ConstableDesc.

In JAVA12 versions, the classes like String, Integer, Byte, Float,.... are implementing these two interfaces and they are providing the following two methods.

1. public Optional describeConstaable()
2. public String resolveConstable()

Both the mnethodfs are representing their objects themselves.
EX:

```
package com.durgasoft.java12features;

public class Test {
        public static void main(String[] args)throws Exception {
                String str = "Durga Software Solutions";
                System.out.println(str.describeConstable().get());
                System.out.println(str.resolveConstantDesc(null));
        }
}
```
OP:

```
Durga Software Solutions
Durga Software Solutions
```

7. Pattern Matching for instanceof operator

In JAVA12 version, it is preview featire, its original implementation we will see in Java14 version.

In general, instanceof operator will check whether a reference variable is representing an instance of a particular class or not.
EX:

```
package java12features;
import java.util.List;
public class Test {
        public static void main(String[] args)throws Exception {
                Object obj = List.of(1,2,3,4,5);
                if(obj instanceof List) {
                        for(int x: (List<Integer>)obj) {
                                System.out.println(x);
```

```
                                System.out.println(x);
                        }
                }else {
                        System.out.println("Content is not matched");
                }
        }
}
```

IN the above example, if we want to do any manipulation with obj then we must convert that obj to List type then only it is possible.

IN JAVA12 version, it is not required to perform type casting directly we are able to get reference variable to use .
EX:

```
package java12features;
import java.util.List;
public class Test {
        public static void main(String[] args)throws Exception {
                Object obj = List.of(1,2,3,4,5);
                if(obj instanceof List list) {
                        for(int x: list) {
                                System.out.println(x);
                        }
                }else {
                        System.out.println("Content is not matched");
                }
        }
}
```

Note: It will not run in JAVA12 version, it will execute in JAVA14 version.

8. Raw String Literals is Removed From JDK 12.

It is a preview feature, it will not be executed in JAVA12 version.
Prior to JAVA12:
```
String html = "<html>\n" +
              "    <body>\n" +
              "             <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

In JAVA12 , we can remove this raw strings, in place of this we will write like below.
```
String html = `<html>
                   <body>
                       <p>Hello World.</p>
                   </body>
               </html>
              `;
```