Java 11 Version Updations:
---
1. Running Java File with single command
2. New utility methods in String class
3. Local-Variable Syntax for Lambda Parameters
4. Nested Based Access Control
5. HTTP Client
6. Reading/Writing Strings to and from the Files
7. Flight Recorder

1. Running Java File with single command:

Upto Java 10 version, if we want to run Java file, first we have to compile Java file with "javac" command then we have to execute Java application by using "java" command.

Java11 version has provided an environment to execute Java file with out compiling java file explicitly, when we execute Java file internally Java file will be compiled , it will

EX:
D:\java11practice\Test.java

```
class Test
{
        public static void main(String[] args)
        {
                System.out.println("Welcome To Java 11 Version");
        }
}
```

On Command Prompt:

```
D:\java11practice>java Test.java
Welcome To Java 11 Version
```

If we provide more than one class with main() method in single java file and if we execute that Java file then JVM will search for the main() method class in the order in which we have provided from starting point of the file to ending point, in which class JVM identifies main() method first time then JVM will treat that class as the actual main class and JVM will execute that main class.
EX:
---
abc.java

```
class A{
  public static void main(String[] args){
     System.out.println("A-main()");
  }
}
class B{
  public static void main(String[] args){
     System.out.println("B-main()");
  }
}
class C{
  public static void main(String[] args){
     System.out.println("C-main()");
  }
```

```
}

D:\java7\java11Features>java abc.java
OP: A-main()

EX:
----
abc.java
---------
class B{
  public static void main(String[] args){
     System.out.println("B-main()");
  }
}
class C{
  public static void main(String[] args){
     System.out.println("C-main()");
  }
}
class A{
  public static void main(String[] args){
     System.out.println("A-main()");
  }
}

D:\java7\java11Features>java abc.java
OP: B-main()

abc.java
---------
class C{
  public static void main(String[] args){
     System.out.println("C-main()");
  }
}
class A{
  public static void main(String[] args){
     System.out.println("A-main()");
  }
}
class B{
  public static void main(String[] args){
     System.out.println("B-main()");
  }
}

D:\java7\java11Features>java abc.java
OP: C-main()

Note: Java11 Version is providing backward Compatibility, that is, in Java11
version, we can follow the old convention like compilation of Java file with
javac and execution of the Java program with java command.
EX:
----
public class Test
{
        public static void main(String[] args)
        {
                System.out.println("Welcome To Java11 Programming");
        }
}
```

```
D:\java7\java11Features>javac Test.java
D:\java7\java11Features>java Test
OP: Welcome To Java11 Programming

2. New utility methods in String class
----------------------------------------
In Java11 version, String class has provided the following new methods.
1. public String releat(int count)
2. public boolean isBlank()
3. public String strip()
4. public String stripLeading()
5. public String stripTrailing()
6. public Stream lines()

1. public String repeat(int count)
--> It will repeate the String object content upto the given no of times.
--> If we provide -ve value then repeat() method will raise an Exception
like     "java.lang.IllegalArgumentException".
--> If the count value is 0 then it will return an empty string.
EX:
---
String str = "Durga";
String str1 = str.repeat(5);
System.out.println(str1);
OP: DurgaDurgaDurgaDurgaDurga

EX:
---
String str = "Durga";
String str1 = str.repeat(0);
System.out.println(str1);
OP: Empty String, No Output

EX:
---
String str = "Durga";
String str1 = str.repeat(-1);
System.out.println(str1);
OP: java.lang.IllegalArgumentException

2. public boolean isBlank()
--> It will check whether the String is Blank[Or White Spaces] or not, if
the String is blank then it will return true value otherwise false value.
EX:
---
String str = "";
System.out.println(str.isBlank());
OP: true
EX:
---
String str = "    ";
System.out.println(str.isBlank());
OP: true
EX:
---
String str = "abc";
System.out.println(str.isBlank());
OP: false

3. public String strip()
```

-->It is same as trim() method, it will remove all leading spaces and trailing spaces of a String.
EX:

```
String str = new String("   Durga Software Solutions   ");
String str1 = str.strip();
System.out.println(str1);
```
OP: Durga Software Solutions

4. public String stripLeading()
--> It will remove all leading spaces[Pre Spaces] to a String, it will not remove Trailing spaces.
EX:

```
String str = new String("   Durga Software Solutions   ");
String str1 = str.stripLeading();
System.out.println(str1);
```
OP: Durga Software Solutions [Here Trailing spaces are existed as it is].

5. public String stripTrailing()
--> It will remove all trailing Spaces to a String, it will not remove leading spaces.
EX:

```
String str = new String("   Durga Software Solutions   ");
String str1 = str.stripTrailing();
System.out.println(str1);
```

6.public Stream lines()
--> This method will split a String into no of Tokens in the form of Stream object on the basis of '\n' literals.
EX:

```
import java.util.stream.*;
import java.util.*;
class Test
{
        public static void main(String[] args) throws Exception
        {
                String str = new String("Durga\nSoftware\nSolutions");
                Stream<String> s = str.lines();
                List<String> l = s.collect(Collectors.toList());
                System.out.println(l);
        }
}
```

3. Local-Variable Syntax for Lambda Parameters:

In Java10 version, Local variables type inference was inroduced, with this, we can declare variables with out providing data type explicitly and just by using "var", here type will be calculated on the basis of the provided variable values.
EX:

```
var str = "Durga";
var i = 10;
```

In Java10 version, "var" is not allowed for the parameter variables.
EX:

```
interface I
```

```
{
        int m1(int i, int j);
}
class Test
{
        public static void main(String[] args)
        {
                I l = (var i,var j) -> i+j;
                System.out.println(l.m1(10,20));
        }
}
```

On Command Prompt:

```
D:\java10>javac Test.java
Test.java:9: error: 'var' is not allowed here
                I l = (var i,var j) -> i+j;
                       ^
Test.java:9: error: 'var' is not allowed here
                I l = (var i,var j) -> i+j;
```

JAVA11 version has provided flexibility to use "var" for the parameter variables in Lambda Expressions.

EX:

```
package java11features;

interface Calculator{
        int add(int i, int j);
}
public class Test {
        public static void main(String[] args) {
                Calculator cal = (var i, var j) -> i+j;
                System.out.println(cal.add(10, 20));
                System.out.println(cal.add(30, 40));

        }
}
```

OP:

```
30
70
```

Limitations:
---

1. We must provide var for all Lambda parameters, not for few lambda parameters.

EX:

```
interface Calculator{
        int add(int i, int j);
}
Calculator cal = (var i, j) -> i+j ----> Invalid
Calculator cal = (var i, int j) -> i+j; --> Invalid
```

2. If we have simple Parameter in Labmda and if we want to use "var" then () are mandatory.

EX:

```
package java11features;
```

```
interface Calculator{
	int sqr(int i);
}
public class Test {
	public static void main(String[] args) {
		//Calculator cal = var i -> i*i; --> invalid
		Calculator cal = (var i) -> i*i;
		System.out.println(cal.sqr(10));
		System.out.println(cal.sqr(30));

	}
}
```

OP:

```
100
900
```

4. Nested Based Access Control:

Upto Java10 version, in inner classes, if we access private member of any inner class in the respective outer class by using reflection API then we are able to get an exception like java.lang.IllegalAccessException, but, if we use Java11 version then we will get any exception , because, in JAVA11 version new methods are introduced in java.lang.Class class like

1. public Class getNestHost() : It able to get the enclosed outer class java.lang.Class object.
2. public Class[] getNestMembers(): It will return all nested members in the form of Class[].
3. public boolean isNestmateOf(): It will check whether a member is nested member or not.

Note: In Java11 the above methods will be executed internally when we access private members of the outer class from nested class and nested members from outer class.

EX:

```
package java11features;

public class Test {
	class NestTest {
		private int count = 10;
	}

	public static void main(String[] args) throws Exception {
		Test.NestTest tt = new Test().new NestTest();
		System.out.println(tt.count);

		java.lang.reflect.Field f =
NestTest.class.getDeclaredField("count");
		f.setInt(tt, 2000);
		System.out.println(tt.count);
	}
}
```

If we run the above program in JAVA10 version then we will get java.lang.IllegalAccessException.

If we run the above program in JAVA11 version we will get out put
10
2000

```
D:\java11practice>set path=C:\Java\jdk-10.0.2\bin;
D:\java11practice>javac Test.java
D:\java11practice>java Test
10
Exception in thread "main" java.lang.IllegalAccessException: class Test
cannot access a member of class Test$NestTest with modifiers "private"

D:\java11practice>set path=C:\Java\jdk11\jdk-11\bin;
D:\java11practice>javac Test.java
D:\java11practice>java Test
10
2000
```

HTTP Client:

Http Client is an API, it can be used to send requests to server side appl or remote appl and to recieve response from Server side appl or Remote applications.

Initially, Http Client API was introduced in JAVA9 version, where it was introduced in a module incubator and it was not fully implemented and it was not in streamlined to use that in Java applications.

JAVA11 version has provided stream lined and standardised implementation for Http Client API in the form of a seperate package "java.net.http".

java.net.http contains mainly three libraries.
1. HttpClient
2. HttpRequest
3. HttpResponse

1. HttpClient:
--> It rerpesents a client for Http protocol, it is responsible to send requests and to recieve response from server.

```
    EX: HttpClient client = HttpClient.newHttpClient();
```

2. HttpRequest:
--> It able to manage request details like url of the Request, type of request, timeout configurations.....

```
    HttpRequest request = HttpRequest.newBuilder()
                          .GET()
                          .uri(URI.create("http://www.durgasoft.com"))
                          .build();
```

3. HttpResponse
--> With the above configurations , if we submit request to Server side appl , Server will generate some response to client , where at client we have to represent response, for representing response we have to use "HttpResponse".
--> HttpResponse is able to provide response details like status code, Response Headers, Response Body
---> HttpResponse contains
     a)statusCode(): It will return status code of the present response.
     b)headers(): It will return headers
     c)body(): It will return the actual response

```
EX:  HttpResponse response = client.send(request);
     System.out.println(response.statusCode());
```

```
     System.out.println(response.headers());
     System.out.println(response.body());
```

EX:

```
package java11features;


import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;


import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;


public class Test {

        public static void main(String[] args) throws Exception {
                var client = HttpClient.newHttpClient();
            var request = HttpRequest.newBuilder()
                        .GET()
                        .uri(URI.create("https://www.google.com"))
                        .build();

              HttpResponse<String> response = client.send(request,
BodyHandlers.ofString());
              System.out.println("Status Code  :  "+response.statusCode());
              System.out.println();
              System.out.println("Response Headers : "+response.headers());
              System.out.println();
              System.out.println("Response Body  : "+response.body());
        }
}
```

If we want to send Post request with form data then we have to use POST() method inplace of GET() and provide request parameters data along with URL like below

```
   http://localhost:1010/loginapp/login?uname=durga&upwd=abc
```

EX:

```
---
Client.java
------------
package com.durgasoft.java11features;

import java.io.BufferedReader;
import java.io.Console;
import java.io.InputStreamReader;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Test {
    public static void main(String[] args)throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("User Name      : ");
        String uname = br.readLine();
```

```
        System.out.print("User Password  : ");
        String upwd = br.readLine();

        var client = HttpClient.newHttpClient();
        var request =
HttpRequest.newBuilder().POST(HttpRequest.BodyPublishers.ofString("")).uri(U
RI.create("http://localhost:1010/loginapp/login?
uname="+uname+"&upwd="+upwd)).build();
        var response = client.send((HttpRequest) request,
HttpResponse.BodyHandlers.ofString());
        //System.out.println("Status Code      : "+response.statusCode());
        //System.out.println();
        //System.out.println("Response Headers : "+response.headers());
        //System.out.println();
        System.out.println("Login Status    : "+response.body());
    }
}
```

Server Side Application:

```
C:\tomcat\webapps
-------------------
loginapp
  |---WEB-INF
         |----classes
                |----LoginServlet.java
                |----LoginServlet.class
```

LoginServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;
@WebServlet("/login")
public class LoginServlet extends HttpServlet
{
        public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException
        {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String uname = request.getParameter("uname");
                String upwd = request.getParameter("upwd");
                System.out.println("uname :"+uname);
                System.out.println("upwd  :"+upwd);
                String status = "";
                if(uname.equals("durga") && upwd.equals("durga"))
                {
                        status = "User Login Success";
                }
                else
                {
                        status = "User Login Failure";
                }
                out.println("<html><body>");
                out.println("<h1>"+status+"</h1>");
                out.println("</body></html>");
        }
}
```

Reading/Writing Strings to and from the Files:

Java11 version has provided the following four new methods in java.nio.file.Files class to read data from file and to write data to file.

1. public static Path writeString(Path path, CharSequence csq, OpenOption... options) throws IOException

2. public static Path writeString(Path path, CharSequence csq, Charset cs, OpenOption... options) throws IOException

3. public static String readString(Path path) throws IOException

3. public static String readString(Path path, Charset cs) throws IOException

EX:

```
package java11features;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.nio.file.Files;


public class Test {
        public static void main(String[] args)throws Exception {
                Path filePath = Paths.get("E:/", "abc/xyz", "welcome.txt");
                Files.writeString(filePath, "Welcome to Durga Software
Solutions");
                String content = Files.readString(filePath);
                System.out.println(content);

        }
}
```

If we want to perform appemd operation in the file then we have to use StandardOpenOption.APPEND constant like below.
Files.writeString(filePath, "Welcome to Durga Software Solutions", StandardOpenOption.APPEND);

Flight Recorder:

Java Flight Recorder is a profiling tool used to gather diagnostics and profiling data from a running Java application, it is available for only for Paying Users, not for normal Users.