INDEX

Java 10 Updations

1) Local-Variable Type Inference [286]
2) Consolidate the JDK Forest into a Single Repository [296]
3) Garbage-Collector Interface [304]
4) Parallel Full GC for G1 [307]
5) Application Class-Data Sharing [310]
6) Thread-Local Handshakes [312]
7) Remove the Native-Header Generation Tool (javah) [313]
8) Additional Unicode Language-Tag Extensions [314]
9) Heap Allocation on Alternative Memory Devices [316]
10) Experimental Java-Based JIT Compiler [317]
11) Root Certificates [319]
12) Time-Based Release Versioning [322]

1) Local-Variable Type Inference:

Upto JAVA9 version, if we want to declare and initialize a variable then we have to use the folloiwing syntax.

```
Data_Type varName = Value;
```

EX:

```
int i = 10;
String str = "abc";
ArrayList<String> al = new ArrayList<String>();
---
----
```

Java10 version has given an option to declare variables with out specifying its type and calculating variable type on the basis of its value , this feature is called as "Type Inference".

In Java10 version, if we want to declare variables with type inference we must use "var" just before variable.

Syntax:

```
var varName = Value;
```

EX:

```
1)  package java10features;
2)  public class Test {
3)
4)     public static void main(String[] args) {
5)        //Upto Java9 version
6)        int i = 10;
7)              String str = "Durga";
8)              System.out.println(i);
9)              System.out.println(str);
10)
11)             // Java 10 version
12)       var j = 10;
13)       System.out.println(j);
14)       var string = "Durga";
15)       System.out.println(string);
16)    }
17) }
```

OUTPUT

```
10
Durga
10
Durga
```

We can use Type Inference to the variables inside the loops also.

EX:

```
1)  package java10features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        //Up to Java9 version
7)        for(int i = 0; i < 10; i++) {
8)           System.out.println(i);
9)        }
10)
11)       // Java10 version
12)       System.out.println();
13)       for(var i = 0; i < 10; i++) {
14)          System.out.println(i);
15)       }
```

```
16)    }
17) }
```

OUTPUT

```
0
1
2
3
4
5
6
7
8
9

0
1
2
3
4
5
6
7
8
9
```

If we want to declare arrays with Type Inference then we must use the following syntax.
var varName = new DataTpe[]{values};

EX:

```
1)  package java10features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        // Up to Java9 version
7)        int[] a = {1,2,3,4,5};
8)        for(int i = 0; i < a.length; i++) {
9)           System.out.println(a[i]);
10)       }
11)       System.out.println();
12)       String[] str = {"AAA", "BBB", "CCC", "DDD", "EEE"};
13)       for(String s: str) {
14)          System.out.println(s);
```

```
15)     }
16)     System.out.println();
17)
18)     //Java10 version
19)     //var x = {1,2,3,4,5}; --> Error
20)     //var string = {"AAA","BBB","CCC","DDD","EEE"}; ---> Error
21)     var x = new int[] {1,2,3,4,5};
22)     for(var i = 0; i < x.length; i++) {
23)        System.out.println(x[i]);
24)     }
25)     System.out.println();
26)     var string = new String[] {"AAA", "BBB", "CCC", "DDD", "EEE"};
27)     for(var element: string ) {
28)        System.out.println(element);
29)     }
30)  }
31) }
```

OUTPUT

```
1
2
3
4
5

AAA
BBB
CCC
DDD
EEE

1
2
3
4
5

AAA
BBB
CCC
DDD
EEE
```

We can use Type Inference for Collection variables declarations.

EX:

```
1)  package java10features;
2)
3)  import java.util.ArrayList;
4)  import java.util.List;
5)
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        // Up to Java9 version
10)       List<String> list1 = new ArrayList<String>();
11)       list1.add("AAA");
12)       list1.add("BBB");
13)       list1.add("CCC");
14)       list1.add("DDD");
15)       System.out.println(list1);
16)
17)       // Java 10 version
18)       var list2 = new ArrayList<String>();
19)       list2.add("AAA");
20)       list2.add("BBB");
21)       list2.add("CCC");
22)       list2.add("DDD");
23)       System.out.println(list2);
24)    }
25) }
```

OUTPUT

[AAA, BBB, CCC, DDD]
[AAA, BBB, CCC, DDD]

Note: In Collection variables declaration we can use diamond operator along with Type Inference, in this context, Compiler will assign Object type to the respective variable , we can add any type of element to the respective Collection.

EX:

```
1)  package java10features;
2)
3)  import java.util.ArrayList;
4)  import java.util.List;
5)
```

```
6)  public class Test {
7)
8)     public static void main(String[] args) {
9)        var list = new ArrayList<>();
10)       list.add(new Integer(10));
11)       list.add("ABC");
12)       list.add(new StringBuffer("XYZ"));
13)       System.out.println(list);
14)    }
15) }
```

Limitations of Type Inference:

1. Type Inference is applicable for only local variables only, not applicable for class level variables.

EX:

```
1)  class A{
2)     int i = 10; --> Valid
3)     var str = "Durga"; ---> Invalid
4)     void m1() {
5)      var j = 20;-----> Valid
6)      System.out.println(j);
7)     }
8)  }
```

2. Type Inference is not possible for Parameters in methods, constructors,....

EX:

```
1)  class A{
2)     A(var i){ ---> Error
3)     }
4)     void m1(var i) { ----> Error
5)     }
6)  }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

3. Type inference is not possible for variables declaration with out initialization and with null value initialization.

EX:

```
1)  class A{
2)    void m1() {
3)       var i; -----> Error
4)       i = 10;
5)
6)       var date = null; ---> Error
7)    }
8)  }
```

4. Type inference is not possible for Lambda Expression variable declaration, but, we can apply Type Inference for variables inside Lambda expressions.

EX:

```
1)  package java10features;
2)
3)  import java.util.function.Predicate;
4)
5)  public class Test {
6)     public static void main(String[] args) {
7)        // Up to Java9 version
8)        Predicate<Integer> p = n -> n < 10;
9)        System.out.println(p.test(5));
10)       System.out.println(p.test(15));
11)       // Java 10 version
12)       var p1 = n -> n < 10; ---> Error
13)    }
14) }
```

EX:

```
1)  package java10features;
2)
3)  import java.util.function.Function;
4)  import java.util.function.Predicate;
5)
6)  public class Test {
7)     public static void main(String[] args) {
8)        Function<Integer, Integer> f = n -> {
9)           var result = n * n;
```

```
10)         return result;
11)      };
12)      System.out.println(f.apply(10));
13)      System.out.println(f.apply(20));
14)   }
15) }
```

OUTPUT

100

400

Note: **In Java 10 version, we can use 'var' as variable name , it will not give any error.**

EX:

```
1)  public class Test {
2)     public static void main(String[] args) {
3)        var var = 10;
4)        System.out.println(var);
5)     }
6)  }
```

OUTPUT

10

API Changes in List, Set and Map:

In Java 10 version, copyOf() function is added in List, Set and Map inorder to get Unmodifiable List, Set and Map.

EX1:

```
1)  package java10features;
2)
3)  import java.util.ArrayList;
4)  import java.util.List;
5)
6)  public class Test {
7)     public static void main(String[] args) {
8)        List<Integer> list = new ArrayList<Integer>();
9)        list.add(10);
10)       list.add(20);
11)       list.add(30);
12)       list.add(40);
13)       System.out.println(list);
```

```
14)      var newList = List.copyOf(list);
15)      System.out.println(newList);
16)      newList.add(60);---> java.lang.UnsupportedOperationException
17)   }
18) }
```

EX:

```
1)  package java10features;
2)  import java.util.HashSet;
3)  import java.util.Set;
4)  public class Test {
5)     public static void main(String[] args) {
6)        Set<Integer> set = new HashSet<Integer>();
7)        set.add(10);
8)        set.add(20);
9)        set.add(30);
10)       set.add(40);
11)       System.out.println(set);
12)       var newSet = Set.copyOf(set);
13)       System.out.println(newSet);
14)       newSet.add(60); ---> java.lang.UnsupportedOperationException
15)   }
16) }
```

API Changes in Collectors Class in Streem API:

Upto JAVA9 version, Collectors class has toList(), toSet(), toMap() methods to get list or set or Map of elements from Streams.

JAVA10 has provided the following methods to Collectors class to provide Unmodifiable or immutable List, Set and Map.

1) **public static List unmodifiableList()**
2) **public static Set unmodifiableSet()**
3) **public static Map unmodifiableMap()**

EX:

```
1)  package java10features;
2)
3)  import java.util.ArrayList;
4)  import java.util.List;
5)  import java.util.stream.Collectors;
6)  import java.util.stream.Stream;
7)
```

```
8)  public class Test {
9)     public static void main(String[] args) {
10)       List<Integer> list = new ArrayList<Integer>();
11)       list.add(5);
12)       list.add(10);
13)       list.add(15);
14)       list.add(20);
15)       list.add(25);
16)       list.add(30);
17)       System.out.println(list);
18)       Stream<Integer> stream = list.stream();
19)       List<Integer> list1 = stream.filter(n-> n%2==0).
    collect(Collectors.toUnmodifiableList());
20)       System.out.println(list1);
21)       list1.add(40); --> java.lang.UnsupportedOperationException
22)    }
23) }
```

EX:

```
1)  package java10features;
2)
3)  import java.util.Set;
4)  import java.util.stream.Collectors;
5)  import java.util.stream.Stream;
6)
7)  public class Test {
8)     public static void main(String[] args) {
9)        Set<Integer> set = Set.of(5,10,15,20,25,30);
10)       Stream<Integer> stream = set.stream();
11)       Set<Integer> s = stream.filter(n->n%2!=0).
    collect(Collectors.toUnmodifiableSet());
12)       System.out.println(s);
13)       s.add(35);  --> java.lang.UnsupportedOperationException
14)    }
15) }
```

2) Consolidate the JDK Forest into a Single Repository:

Upto Java 9 version, the complete java has devided into repositories like root corba, hotspot, jdk, langtools,jaxp, jaxw, nashorn but, in JAVA10 version all these repositories are provided into single Repository under src inorder to simplify and streamline Development.

3) Garbage-Collector Interface

Upto Java9 version, Different Garbage Collectors are existed in Java like G1, CMS,... which are available in different modules in java, they are not providing cleaner implementation, but, JAVA10 version has introduced GC interface for cleaner implementation of Garbage Collectors.

4) Parallel Full GC for G1

In Java9 version, G1 Garbage is Default Garbage Collector, it will provide effective perofrmance on Single Threadding, when it comes for Concurrent collection its perofrmance is down, to overcome this problem, JAVA 10 has provided Multi Threadded model with G1 Garbage Collector in the form of Parallel Full GC.

5) Application Class-Data Sharing:

The main intention of Class-Data Sharing is to share loaded class to multiple JVMs through shared memory when we are working large scale applications, Class-Data Sharing was introduced in JSD1.5 version and it was limited to System classes and Serial GC, but, JAVA10 version, it was extended to allow application classes along with SYstem clsses and Garbage Collectors.

6) Thread-Local Handshakes:

- When we Lock an object by a thread explicitly or when we apply synchronization on objects then other threa will come to blocked state safely whose state is described by other threads in JVM , here threads which are in safepoint. In Some Situation JVM will keep all the Threads in Blocked state called as "Stop The Word", before Stop the Word , all threads will come to safepoint, this safepoint is called as Global Safepoint.

- A handshake operation is a callback that is executed for each Java Thread while that thread is in a safepoint state. The callback is executed either by the thread itself or by the VM thread while keeping the thread in a blocked state.

- Thread-Local Handshakes is JVM internal feature to improve performance, This feature provides a way to execute a callback on threads without performing a global VM safepoint.

7) Remove the Native-Header Generation Tool (javah):

- javah is a C-Header file generator from Java classes, it will be used in Java Native Interfaces which accessing native Methods.

- JAVA10 version has removed javah from Java software, because, javah capabilities are already embedded with javac command in JAVA8 version.

8) Additional Unicode Language-Tag Extensions

Up to Java9 version, java.lang.Locale has the UNICODE support for "ca" [Calender] and "nu"[Numeric], but, JAVA10 version has given UNICODE support for "cu"[Currency], "fw"[First Day Of Week], "rg" [Region Override], "tz"[Time Zone].

9) Heap Allocation on Alternative Memory Devices

- This featur allows the HotSpot VM to allocate the Java object heap on an alternative memory device, specified by the user.

- It is very much required in Multi JVM environment, in BIG Data applications, we need to execute some low priority threads like Daemon threads, Services threads and some other High priority threads,

- This new feature would make it possible in a multi-JVM environment to assign lower priority processes to use the NV-DIMM memory, and instead only allocate the higher priority processes to the DRAM.

- NV-DIMM → Non Volatile Dual In Memory Modules, it is low cost and having larger capacity.
- DRAM → Dynamic Random Access Memory, High Cost and More Potential Memory.

10) Experimental Java-Based JIT Compiler:

- In JVM, JIT Compiler will increase perofrmance of JVM while converting Byte code to Native, but, it was written in C++ .
- JAVA10 has provided an experimental JIT Compiler written in Java completely and it was provided in support of Linux/64 bit OS.

11) Root Certificates

It is Security feature in Java, It is Providing root CA[Certificate Authority] certificates makes "OpenJDK builds more attractive to developers" and "reduces [sic] the differences between those builds and Oracle JDK builds".

12) Time-Based Release Versioning

- **Starting from Java 10, Oracle has adapted time based version-string scheme [JEP 322]. The new time-based model has replaced the feature-based, multi-year release model of the past. Unlike the old releases, the new time-based releases will not be delayed and features will be released every six months, with no constraints on what features can go out in the releases.**

- **The updates releases will occur every quarter (Jan, April, July, Oct). Update releases will be strictly limited to fixes of security issues, regressions, and bugs in newer features. Going by schedule planning, we can say that each feature release will receive two updates before the next feature release.**

- **The new version format of Java is $FEATURE.$INTERIM.$UPDATE.$PATCH**

- **$FEATURE : It will be incremented every 6 months and based on feature release versions e.g: JDK 10, JDK 11.**

- **$INTERIM : It will be incremented for non-feature releases that contain compatible bug fixes and enhancements.**

- **$UPDATE : It will be incremented for compatible update releases that fix security issues, regressions, and bugs in newer features.**

- **$PATCH : It will be incremented only when it's necessary to produce an emergency release to fix a critical issue.**

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

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        System.out.println("Welcome To Java 11 Version");
6)     }
7)  }
```

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

```
1)  class A{
2)    public static void main(String[] args){
3)      System.out.println("A-main()");
4)    }
5)  }
6)  class B{
7)    public static void main(String[] args){
8)      System.out.println("B-main()");
9)    }
10) }
11) class C{
12)   public static void main(String[] args){
13)     System.out.println("C-main()");
14)   }
15) }
```

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

EX: abc.java

```
1)  class B {
2)    public static void main(String[] args){
3)      System.out.println("B-main()");
4)    }
5)  }
6)  class C{
7)    public static void main(String[] args){
8)      System.out.println("C-main()");
9)    }
10) }
11) class A{
12)   public static void main(String[] args){
13)     System.out.println("A-main()");
14)   }
15) }
```

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

abc.java

```
1)  class C{
2)    public static void main(String[] args){
3)      System.out.println("C-main()");
4)    }
5)  }
6)  class A{
7)    public static void main(String[] args){
8)      System.out.println("A-main()");
9)    }
10) }
11) class B{
12)   public static void main(String[] args){
13)     System.out.println("B-main()");
14)   }
15) }
```

D:\java7\java11Features>java abc.java
OUTPUT 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:

```
1)  public class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        System.out.println("Welcome To Java11 Programming");
6)     }
7)  }
```

D:\java7\java11Features>javac Test.java
D:\java7\java11Features>java Test
OUTPUT 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 repeat(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);
OUTPUT DurgaDurgaDurgaDurgaDurga
```

EX:

```
String str = "Durga";
String str1 = str.repeat(0);
System.out.println(str1);
OUTPUT Empty String, No Output
```

EX:

```
String str = "Durga";
String str1 = str.repeat(-1);
System.out.println(str1);
OUTPUT java.lang.IllegalArgumentException
```

1) 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());
OUTPUT true
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

EX:

```
String str = "    ";
System.out.println(str.isBlank());
```

OUTPUT true

EX:

```
String str = "abc";
System.out.println(str.isBlank());
```

OUTPUT false

2) 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);
```

OUTPUT Durga Software Solutions

3) 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);
```

OUTPUT Durga Software Solutions [Here Trailing spaces are existed as it is].

4) 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);
```

5) 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:

```
1)  import java.util.stream.*;
2)  import java.util.*;
3)  class Test
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String str = new String("Durga\nSoftware\nSolutions");
8)        Stream<String> s = str.lines();
9)        List<String> l = s.collect(Collectors.toList());
10)       System.out.println(l);
11)    }
12) }
```

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:

```
1)  interface I
2)  {
3)     int m1(int i, int j);
4)  }
5)  class Test
6)  {
7)     public static void main(String[] args)
8)     {
9)        I l = (var i,var j) -> i+j;
10)       System.out.println(l.m1(10,20));
11)    }
12) }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

On Command Prompt:

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

JAVA 11 version has provided flexibility to use "var" for the parameter variables in Lambda Expressions.

EX:

```
1)  package java11features;
2)
3)  interface Calculator{
4)     int add(int i, int j);
5)  }
6)  public class Test {
7)     public static void main(String[] args) {
8)        Calculator cal = (var i, var j) -> i+j;
9)        System.out.println(cal.add(10, 20));
10)       System.out.println(cal.add(30, 40));
11)
12)    }
13) }
```

OUTPUT

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
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

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

EX:

```
1)  package java11features;
2)
3)  interface Calculator{
4)     int sqr(int i);
5)  }
6)  public class Test {
7)     public static void main(String[] args) {
8)              //Calculator cal = var i -> i*i; --> invalid
9)        Calculator cal = (var i) -> i*i;
10)       System.out.println(cal.sqr(10));
11)       System.out.println(cal.sqr(30));
12)
13)    }
14) }
```

OUTPUT
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.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

EX:

```
1)  package java11features;
2)
3)  public class Test {
4)     class NestTest {
5)        private int count = 10;
6)     }
7)
8)     public static void main(String[] args) throws Exception {
9)        Test.NestTest tt = new Test().new NestTest();
10)       System.out.println(tt.count);
11)
12)       java.lang.reflect.Field f = NestTest.class.getDeclaredField("count");
13)       f.setInt(tt, 2000);
14)       System.out.println(tt.count);
15)    }
16) }
```

If we run the above program then we will get java.lang.IllegalAccessException.
If we run the above program 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 3 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());
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

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

EX:

```
1)  package java11features;
2)
3)  import java.net.URI;
4)  import java.net.http.HttpClient;
5)  import java.net.http.HttpRequest;
6)
7)  import java.net.http.HttpResponse;
8)  import java.net.http.HttpResponse.BodyHandlers;
9)
10) public class Test {
11)
12)    public static void main(String[] args) throws Exception {
13)       var client = HttpClient.newHttpClient();
14)       var request = HttpRequest.newBuilder()
15)                   .GET()
16)                   .uri(URI.create("https://www.google.com"))
17)                   .build();
18)
19)     HttpResponse<String> response = client.send(request, BodyHandlers.ofString());

20)         System.out.println("Status Code  :  "+response.statusCode());
21)         System.out.println();
22)         System.out.println("Response Headers : "+response.headers());
23)         System.out.println();
24)         System.out.println("Response Body  : "+response.body());
25)    }
26) }
```

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

```
1)  package com.durgasoft.java11features;
2)
3)  import java.io.BufferedReader;
4)  import java.io.Console;
5)  import java.io.InputStreamReader;
6)  import java.net.URI;
7)  import java.net.http.HttpClient;
```

```
8)  import java.net.http.HttpRequest;
9)  import java.net.http.HttpResponse;
10)
11) public class Test {
12)    public static void main(String[] args)throws Exception {
13)      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

14)       System.out.print("User Name     : ");
15)       String uname = br.readLine();
16)       System.out.print("User Password  : ");
17)       String upwd = br.readLine();
18)
19)       var client = HttpClient.newHttpClient();
20)       var request = HttpRequest.newBuilder().POST(HttpRequest.BodyPublishers.of
    String("")).uri(URI.create("http://localhost:1010/loginapp/login?uname="+uname
    +"&upwd="+upwd)).build();
21)       var response = client.send((HttpRequest) request, HttpResponse.BodyHandlers
    .ofString());
22)       //System.out.println("Status Code     : "+response.statusCode());
23)       //System.out.println();
24)       //System.out.println("Response Headers : "+response.headers());
25)       //System.out.println();
26)       System.out.println("Login Status    : "+response.body());
27)    }
28) }
```

Server Side Application:

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

LoginServlet.java

```
1)  import java.io.*;
2)  import javax.servlet.*;
3)  import javax.servlet.http.*;
4)  import javax.servlet.annotation.*;
5)  @WebServlet("/login")
6)  public class LoginServlet extends HttpServlet
7)  {
```

```
8)    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
9)    {
10)      response.setContentType("text/html");
11)      PrintWriter out = response.getWriter();
12)      String uname = request.getParameter("uname");
13)      String upwd = request.getParameter("upwd");
14)      System.out.println("uname :"+uname);
15)      System.out.println("upwd  :"+upwd);
16)      String status = "";
17)      if(uname.equals("durga") && upwd.equals("durga"))
18)      {
19)         status = "User Login Success";
20)      }
21)      else
22)      {
23)         status = "User Login Failure";
24)      }
25)      out.println("<html><body>");
26)      out.println("<h1>"+status+"</h1>");
27)      out.println("</body></html>");
28)   }
29) }
```

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:

```
1)  package java11features;
2)  import java.nio.file.Path;
3)  import java.nio.file.Paths;
4)  import java.nio.file.StandardOpenOption;
5)  import java.nio.file.Files;
6)
7)  public class Test {
8)     public static void main(String[] args)throws Exception {
9)        Path filePath = Paths.get("E:/", "abc/xyz", "welcome.txt");
10)       Files.writeString(filePath, "Welcome to Durga Software Solutions");
11)       String content = Files.readString(filePath);
12)       System.out.println(content);
13)    }
14) }
```

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.

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;
    }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

From Java12 version onwards, 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:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        var i = 10;
6)        switch(i){
7)           case 5,10:
8)              System.out.println("Five or Ten");
9)           break;
10)
11)          case 15, 20:
12)             System.out.println("Fifteen or Twenty");
13)          break;
14)
15)          default:
16)             System.out.println("Defasult");
17)          break;
18)       }
19)    }
20) }
```

OUTPUT

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:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        var i = 10;
6)        var result = switch(i){
7)           case 5:
8)             break "Five";
9)                   case 10:
10)                    break "Ten";
11)          case 15:
12)            break "Fifteen";
13)                  case 20:
14)                    break "Twenty";
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

```
15)         default:
16)             break "Number not in 5, 10, 15 and 20";
17)       };
18)       System.out.println(result);
19)    }
20) }
```

OUTPUT

Ten

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

EX:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        var i = 10;
6)        var result = switch(i){
7)           case 5, 10:
8)              break "Five or Ten";
9)           case 15,20:
10)             break "Fifteen or Twenty";
11)
12)          default:
13)             break "Number not in 5, 10, 15 and 20";
14)       };
15)       System.out.println(result);
16)    }
17) }
```

OUTPUT

Five or Ten

2) By using Lambda Style Syntax:

Syntax:

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

EX:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        var i = 10;
6)        var result = switch(i){
7)           case 5 -> "Five";
8)                    case 10 -> "Ten";
9)           case 15 -> "Fifteen";
10)                   case 20 -> "Twenty";
11)          default -> "Number not in 5, 10, 15 and 20";
12)       };
13)       System.out.println(result);
14)    }
15) }
```

OUTPUT

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:

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        var i = 10;
6)        var result = switch(i){
7)           case 5, 10 -> "Five or Ten";
8)           case 15,20 -> "Fifteen or Twenty";
9)           default ->  "Number not in 5, 10, 15 and 20";
10)       };
11)       System.out.println(result);
12)    }
13) }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

OUTPUT

Five or Ten

EX:

```
1)  import java.io.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        var day = "";
7)        var dayType = "";
8)        try{
9)           BufferedReader br = new BufferedReader(new InputStreamReader
    (System.in));
10)          System.out.print("Enter Day [Upper case letters only]  : ");
11)          day = br.readLine();
12)          dayType = switch(day){
13)             case "MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY" ->
    "WEEK DAY";
14)             case "SATURDAY","SUNDAY" -> "WEEKEND";
15)             default -> day+" is an invalid Day";
16)          };
17)       }catch(Exception e){
18)          System.out.println(e.getMessage());
19)       }
20)       System.out.println(day+" is  "+dayType);
21)    }
22) }
```

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:

```
1)  package java12features;
2)
3)  import java.nio.file.Files;
4)  import java.nio.file.Path;
5)  import java.nio.file.Paths;
6)
7)  public class Test {
8)
9)     public static void main(String[] args)throws Exception {
10)
11)       Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
12)       Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
13)       Files.writeString(file1, "Welcome To Durga Software Solutions");
14)       Files.writeString(file2, "Welcome To Durga Software Solutions");
15)       long val = Files.mismatch(file1, file2);
16)       System.out.println(val);
17)       if(val == -1L) {
18)          System.out.println("Files Content is Matched ");
19)       }else {
20)          System.out.println("Files Content isMismatched `");
21)       }
22)    }
23) }
```

OUTPUT

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

EX:

```
1)  package java12features;
2)
3)  import java.nio.file.Files;
4)  import java.nio.file.Path;
5)  import java.nio.file.Paths;
6)
7)  public class Test {
8)
9)     public static void main(String[] args)throws Exception {
10)
11)       Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
12)       Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
13)       Files.writeString(file1, "Welcome To Durga Software Solutions");
14)       Files.writeString(file2, "Welcome To Durgasoft");
15)       long val = Files.mismatch(file1, file2);
16)       System.out.println(val);
17)       if(val == -1L) {
18)          System.out.println("Files Content is Matched ");
19)       }else {
20)          System.out.println("Files Content isMismatched `");
21)       }
22)    }
23) }
```

OUTPUT

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 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:

```
1)  package com.durgasoft.java12features;
2)
3)  import java.text.NumberFormat;
4)  import java.util.Locale;
5)
6)  public class Test {
7)
8)     public static void main(String[] args)throws Exception {
9)        // Converting value from Normal number to Compact Number in Short form
10)       NumberFormat shortForm = NumberFormat.getCompactNumberInstance(new
      Locale("en", "US"),NumberFormat.Style.SHORT);
11)       System.out.println("1000 ----> "+shortForm.format(1000));
12)       System.out.println("10000 ---> "+shortForm.format(10000));
13)       System.out.println("100000 --> "+shortForm.format(100000));
14)       System.out.println();
15)
16)       // Converting value from Normal number to Compact Number in Long form
17)       NumberFormat longForm = NumberFormat.getCompactNumberInstance(new
     Locale("en", "US"),NumberFormat.Style.LONG);
18)       System.out.println("1000 ----> "+longForm.format(1000));
19)       System.out.println("10000 ---> "+longForm.format(10000));
20)       System.out.println("100000 --> "+longForm.format(100000));
```

```
21)
22)       // Converting value from Compact number in Long Form to Normal Number
23)       System.out.println();
24)       NumberFormat numFormat = NumberFormat.getCompactNumberInstance
      (new Locale("en", "US"),NumberFormat.Style.LONG);
25)       System.out.println("1 thousand -----> "+numFormat.parse("1 thousand"));
26)       System.out.println("10 thousand ----> "+numFormat.parse("10 thousand"));
27)       System.out.println("100 thousand ---> "+numFormat.parse("100 thousand"));
28)
29)       // Converting value from Compact number in Short Form to Normal Number
30)       System.out.println();
31)       NumberFormat numFmt = NumberFormat.getCompactNumberInstance
      (new Locale("en", "US"),NumberFormat.Style.SHORT);
32)       System.out.println("1k -----> "+numFmt.parse("1k "));
33)       System.out.println("10k ----> "+numFmt.parse("10k"));
34)       System.out.println("100k ---> "+numFmt.parse("100k"));
35)    }
36) }
```

OUTPUT

```
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:

```
1)  package java12features;
2)
3)  import java.util.stream.Collectors;
4)  import java.util.stream.Stream;
5)
6)  public class Test {
7)
8)     public static void main(String[] args)throws Exception {
9)        double mean = Stream.of(1,2,3,4,5,6,7,8,9,10).collect(
10)            Collectors.teeing(
11)                  Collectors.summingDouble(x->x),
12)                  Collectors.counting(),
13)                  (sum,count)->sum/count));
14)       System.out.println("Mean : "+mean);
15)    }
16) }
```

OUTPUT

Mean : 5.5

5) Java Strings New Methods

JAVA 12 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:

```
1)  package java12features;
2)  public class Test {
3)     public static void main(String[] args)throws Exception {
4)        String str1 = "Durga\nSoftware\nSolutions";
5)        System.out.println(str1);
6)        String newString1 = str1.indent(5);
7)        System.out.println(newString1);
```

```
8)      System.out.println();
9)      String str2 = "    Durga\n    Software\n    Solutions";
10)     System.out.println(str2);
11)     String newString2 = str2.indent(-5);
12)     System.out.println(newString2);
13)     System.out.println();
14)     String str3 = "   Durga\n   Software\n   Solutions";
15)     System.out.println(str3);
16)     String newString3 = str3.indent(-5);
17)     System.out.println(newString3);
18)     System.out.println();
19)     String str4 = "   Durga\n   Software\n   Solutions";
20)     System.out.println(str4);
21)     String newString4 = str4.indent(0);
22)     System.out.println(newString4);
23)   }
24) }
```

OUTPUT

```
Durga
Software
Solutions
    Durga
    Software
    Solutions

    Durga
    Software
    Solutions
Durga
Software
Solutions

  Durga
  Software
  Solutions
Durga
Software
Solutions

  Durga
  Software
  Solutions
  Durga
  Software
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

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:

```
1)  package java12features;
2)  public class Test {
3)     public static void main(String[] args)throws Exception {
4)        String str = "Durga Software Solutions";
5)        String newString = str.transform(s->s.toUpperCase());
6)        System.out.println(newString);
7)     }
8)  }
```

OUTPUT

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:

```
1)  package com.durgasoft.java12features;
2)
3)  public class Test {
4)     public static void main(String[] args)throws Exception {
5)        String str = "Durga Software Solutions";
6)        System.out.println(str.describeConstable().get());
7)        System.out.println(str.resolveConstantDesc(null));
8)     }
9)  }
```

OUTPUT

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:

```
1)  package java12features;
2)  import java.util.List;
3)  public class Test {
4)     public static void main(String[] args)throws Exception {
5)        Object obj = List.of(1,2,3,4,5);
6)        if(obj instanceof List) {
7)           for(int x: (List<Integer>)obj) {
8)              System.out.println(x);
9)           }
10)       }else {
11)          System.out.println("Content is not matched");
12)       }
13)   }
14) }
```

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:

```
1)  package java12features;
2)  import java.util.List;
3)  public class Test {
4)     public static void main(String[] args)throws Exception {
5)        Object obj = List.of(1,2,3,4,5);
6)        if(obj instanceof List list) {
7)           for(int x: list) {
8)              System.out.println(x);
9)           }
```

```
10)       }else {
11)          System.out.println("Content is not matched");
12)       }
13)    }
14) }
```

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>
          `;
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

Java 13 Updations

1) Text Blocks
2) New Methods in String Class for Text Blocks
3) Switch Expressions Enhancements
4) Reimplement the Legacy Socket API
5) Dynamic CDS Archive
6) ZGC: Uncommit Unused Memory
7) FileSystems.newFileSystem() Method

1)Text Blocks:

Upto Java12 version, if we want to write String data in more than one line then we have to use \n chacter in middle of the String.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String address = "\tDurga Softweare Solutions\n"+
7)              "\t202, HMDA, Mitryvanam\n"+
8)              "\tAmeerpet, Hyd-38";
9)        System.out.println(address);
10)    }
11) }
```

OUTPUT

```
Durga Softweare Solutions
202, HMDA, Mitryvanam
Ameerpet, Hyd-38
```

In the above code, to bring string data into new line wqe have to use \n character and to provide space before each and every line we have to use \t, these escape symbols may or may not support by all the operating systems.

To overcome the above problems, JAVA13 version has provided a new preview feature called as "Text Blocks", where we can write String data in multiple lines with out providing \n and \t escape symbopls.

Syntax:

```
String varName = """
                String Data in line1
                String Data in line2
                ----
                """
```

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)      public static void main(String[] args) {
6)          String address = """
7)                  Durga Softweare Solutions
8)                  202, HMDA, Mitryvanam\n
9)                  Ameerpet, Hyd-38""";
10)         System.out.println(address);
11)     }
12)
13) }
```

OUTPUT

```
Durga Softweare Solutions
202, HMDA, Mitryvanam
Ameerpet, Hyd-38
```

Note: If we want to execute preview features we have to enable preview Features first before execution, for thsi we have to use the following approaches.

1. If we want to execute program through Command prompt then use the following commands.

```
javac --enable-preview --release 13 Test.java
java --enable-preview Test
```

2. In Eclipse IDE:
   1) Right Click on Project
   2) Properties
   3) Java Compiler
   4) Uncheck "Use Compliance from Execution Evenvironment JAVASE13.
   5) Select "use --release option"
   6) Unselect "Use default complaince Settings"
   7) Select "Enable preview Features on Java13 version".

--> In Text Block, we have to provide data in the next line after """ and we can provide end """ either in the same line of data or in the next line, Data must not be provided in the same line of """.

EX:

String str1 = """Durga Software Solutions"; --> Invalid.
String str2 = """
Durga Software Solutions"""; ---> Valid
String str3 = """
Durga Software Solutions
"""; ----------> Valid

--> In Java, If we provide same data in normal "" and the data in """ """ then we are able to compare by using equals() and == operators , in both the cases we are able to get true value.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String data = "Durga Software Solutions";
7)        String newData = """
8)              Durga Software Solutions""";
9)        System.out.println(data);
10)       System.out.println(newData);
11)       System.out.println(data.equals(newData));
12)       System.out.println(data == newData);
13)    }
14) }
```

OUTPUT

Durga Software Solutions
Durga Software Solutions
true
true

--> It is possible to perform concatination operation between the strings which areepresented in "" and """ """.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String str1 = "Durga ";
7)        String str2 = """
8)              Softare""";
9)        String str3 = " Solutions";
10)       String result1 = str1+str2+str3;
11)       String result2 = str1.concat(str2).concat(str3);
12)       System.out.println(result1);
13)       System.out.println(result2);
14)    }
15) }
```

OUTPUT

```
Durga Softare Solutions
Durga Softare Solutions
```

--> It is possible to pass Text Blocks as parameters to the methods and it is possible to return Text Blocks from the methods.

EX1:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        System.out.println("""
7)              Durga Software Solutions
8)              202, HMDA, Mitryvanam
9)              Ameerpet,Hyd-38
10)             """);
11)    }
12) }
```

OUTPUT

```
Durga Software Solutions
202, HMDA, Mitryvanam
Ameerpet,Hyd-38
```

EX:

```
1)  package java13features;
2)
3)  class Employee{
4)     public String getEmpDetails(String str) {
5)        String result ="""
6)              Employee Details:
7)              ----------------------
8)              """+str+"""
9)              ESAL   : 15000
10)             EADDR  : Hyd
11)             EEMAIL : durga@dss.com
12)             EMOBILE: 91-9988776655
13)             """;
14)       return result;
15)    }
16) }
17) public class Test {
18)
19)    public static void main(String[] args) {
20)       String str = """
21)             EID    : E-111
22)             ENAME  : Durga
23)             """;
24)       Employee emp = new Employee();
25)       String result = emp.getEmpDetails(str);
26)       System.out.println(result);
27)    }
28) }
```

OUTPUT

```
Employee Details:
----------------------
EID    : E-111
ENAME  : Durga
ESAL   : 15000
EADDR  : Hyd
EEMAIL : durga@dss.com
EMOBILE: 91-9988776655
```

Text Block is very much usefull feature to represent Html or JSON code in Java applications.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String html = """
7)              <html>
8)                 <body>
9)                    <p>Durg Software Solutions</p>.
10)                </body>
11)             </html>
12)             """;
13)       System.out.println(html);
14)    }
15) }
```

OUTPUT

```
<html>
        <body>
                <p>Durg Software Solutions</p>.
        </body>
</html>
```

IN the above Html code, how much spaces are provided at each and every line the same no of spaces will be provided in the output.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String html = """
7)                <html>
8)                   <body>
9)                      <p>Durg Software Solutions</p>.
10)                  </body>
11)               </html>
12)             """;
13)       System.out.println(html);
14)    }
15) }
```

OUTPUT

```
<html>
    <body>
        <p>Durg Software Solutions</p>.
    </body>
</html>
```

--> On Text Blocks we can apply intend() method inorder to mange spaces.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)      public static void main(String[] args) {
6)          String data1 = """
7)                  Durga Software Solutions
8)                  """;
9)          System.out.println(data1);
10)         System.out.println(data1.indent(5));
11)         System.out.println();
12)         String data2 = """
13)                     Durga Software Solutions
14)                 """;
15)         System.out.println(data2);
16)         System.out.println(data2.indent(-5));
17)
18)     }
19) }
```

OUTPUT

Durga Software Solutions

    Durga Software Solutions

    Durga Software Solutions

Durga Software Solutions

--> It is possible to provide ' '[Single qutations] and ""[Double quotations] directly, but, we can provide """[Triple Quotations] with \.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)        String str = """
7)              'Durga Software Solutions'
8)              "Ameerpet"
9)              \"""Hyderabad\"""
10)             """;
11)       System.out.println(str);
12)
13)    }
14) }
```

OUTPUT

```
'Durga Software Solutions'
"Ameerpet"
"""Hyderabad"""
```

2) New Methods in String Class for Text Blocks

For Text Blocks, JAVA13 version has provided the following three methods in String class.

1) public String formatted(Object ... values)
2) public String stripIndent()
3) public Sltring translateEscapes()

1) public String formatted(Object ... values):

In Text Blocks we can provide data formating by using the following method.
public String formatted(val1,val2,...val_n)

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)     public static void main(String[] args) {
6)
7)        String empDetails = """
8)              Employee Details:
9)              ----------------------
```

```
10)             Employee Number   : %d
11)             Employee Name     : %s
12)             Employee Salary   : %f
13)             Employee Address  : %s
14)             """;
15)       System.out.println(empDetails.formatted(111,"Durga",25000.0f,"Hyd"));
16)       System.out.println();
17)       System.out.println(empDetails.formatted(222,"Anil",35000.0f,"Hyd"));
18)       System.out.println();
19)       System.out.println(empDetails.formatted(333,"Ravi",45000.0f,"Hyd"));
20)    }
21) }
```

OUTPUT

```
Employee Details:
----------------------
Employee Number   : 111
Employee Name     : Durga
Employee Salary   : 25000.000000
Employee Address  : Hyd

Employee Details:
----------------------
Employee Number   : 222
Employee Name     : Anil
Employee Salary   : 35000.000000
Employee Address  : Hyd

Employee Details:
----------------------
Employee Number   : 333
Employee Name     : Ravi
Employee Salary   : 45000.000000
Employee Address  : Hyd
```

2) public String stripIndent():

If we have any spces at begining and Ending of String.

EX:

```
1)  package java13features;
2)
3)  public class Test {
4)
5)      public static void main(String[] args) {
6)          String data1 = "\t\tDurga Software Solutions \t\t";
7)          String data2 = "\t\tHyderabad\t\t";
8)          System.out.println(data1+data2);
9)          System.out.println(data1.stripIndent()+data2.stripIndent());
10)
11)     }
12) }
```

OUTPUT

```
                Durga Software Solutions                    Hyderabad
Durga Software SolutionsHyderabad
```

3) public Sltring translateEscapes()

It able to remove Escape symbols like \ in our Srting data.

EX:

```
1)  package java13features;
2)
3)  import java.io.FileInputStream;
4)  import java.io.FileOutputStream;
5)
6)  public class Test {
7)
8)      public static void main(String[] args)throws Exception {
9)
10)         String str1 = "Durga\nSoftware\nSolutions";
11)         String str2 = "Durga\\nSoftware\\nSolutions";
12)         System.out.println(str1);
13)         System.out.println();
14)         System.out.println(str2);
15)         System.out.println();
16)         System.out.println(str2.translateEscapes());
17)     }
```

```
18) }
```

OUTPUT

```
Durga
Software
Solutions

Durga\nSoftware\nSolutions

Durga
Software
Solutions
```

3) Switch Expressions Enhancements:

Switch Expression in JAVA12 was introduced as preview feature, in JAVA13 also it was very same except break statement. In JAVA13 version, we can use "yield" in place of break statement to return a value.

Note:

In JAVA13 version, we are able to use switch as statement and as Expression.
In Switch statement we will use "yield" keyword to return a value in place of break statement.
IN Switch Expression, we will use -> to return value.

EX:

```
1)  public class Hello {
2)     public static void main(String[] args) {
3)        var val = "SUNDAY";
4)        var result = switch (val){
5)           case "MONDAY","TUESDAY","WENSDAY","THURSDAY","FRIDAY":
6)              yield "Week Day";
7)           case "SATURDAY","SUNDAY":
8)              yield "Weekend";
9)           default:
10)             yield  "Not a Day";
11)       };
12)       System.out.println(result);
13)    }
14) }
```

OUTPUT

```
Weekend
```

EX:

```
1)  public class Hello {
2)     public static void main(String[] args) {
3)        var val = "SUNDAY";
4)       var result = switch (val){
5)        case "MONDAY","TUESDAY","WENSDAY","THURSDAY","FRIDAY" ->
     "Week Day";
6)        case "SATURDAY","SUNDAY" -> "Weekend";
7)        default -> "Not a Day";
8)       };
9)        System.out.println(result);
10)    }
11) }
```

OUTPUT

Weekend

4) Reimplement the Legacy Socket API:

Upto JAVA12 version, Socket API contains old library which was provided before JDK1.0 version which uses java.net.Socket and java.net.ServerSocket, JAVA13 version has provided new Implementation for Socket API which is based on NioSocketImpl inplace of PlainSocketImpl.

New Socket API used java.util.concurrent locks rather than synchronized methods.
If you want to use the legacy implementation, use the java option -
Djdk.net.usePlainSocketImpl for backword compatibility.

5) Dynamic CDS Archive:

Class Data Sharing was introduced in JAVA10 version, it was difficult to prepare cds, but, JAVA13 version simplifies CDS creation by using the following command.

java -XX:ArchiveClassesAtExit=my_app_cds.jsa -cp my_app.jar

java -XX:SharedArchiveFile=my_app_cds.jsa -cp my_app.jar

6) ZGC: Uncommit Unused Memory:

Z Garbage Collector was introduced in JAVA11 version, it has provide small span of time before clean up memory and the unused memory was not returned to Operating System, but, JAVA13 version has provided an enhancement over
Z garbage Collector, it will return uncommitted and unused memory to Operating System.

7) FileSystems.newFileSystem() Method

Java1e3 version has provided three new methods have been added to the FileSystems class to make it easier to use file system providers, which treats the contents of a file as a file system.

newFileSystem(Path)
newFileSystem(Path, Map<String, ?>)
newFileSystem(Path, Map<String, ?>, ClassLoader)

JAVA 14 Updations

Developers Required Features:

1) **Switch Expressions (Standard)**
2) **Pattern Matching for instanceof (Preview)**
3) **Helpful NullPointerExceptions**
4) **Records (Preview)**
5) **Text Blocks (Second Preview)**

1) Switch Expressions (Standard):

JAVA12 version has introduced switch expression with break statement and lambda syntaxes, which allows multiple case lebelles.

Syntaxes:

```
var result = switch(value){
  case l1,l2,l3:
    break value;
  ----
  default:
    break value;
}

var result = switch(value){
  case l1,l2,l3 -> value;
  ----
  default -> value;
}
```

In Java 13 version, we must use "yield" keyword inplace of "break" to return value from switch.

```
var result = switch(value){
  case l1,l2,l3:
    yield value;
  ----
  default:
    yield value;
}
```

JAVA14 version made switch statement as standard feature with lambda syntax and "yield" keyword.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

Syntax:

```
var result = switch(value){
 case l1,l2,l3 ->{
      yield value;
 }
 ----
 default ->{
    yield value;
 }
}
```

EX:

```
1)  package java14features;
2)
3)  import java.util.Scanner;
4)
5)  public class Test {
6)     public static void main(String[] args) {
7)        var scanner = new Scanner(System.in);
8)        System.out.print("Enter Day Of WEEK : ");
9)        var value = scanner.next().toUpperCase();
10)       var result = switch (value){
11)          case "MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY" ->
     "Weekday";
12)          case "SATURDAY","SUNDAY" -> "Weekend";
13)          default ->{
14)             if(value.isEmpty()){
15)                yield "value to switch is empty, please check";
16)             }else{
17)                yield value+"""
18)
19)             Value must be in
20)             1.MONDAY
21)             2.TUESDAY
22)             3.WEDNESDAY
23)             4.THURSDAY
24)             5.FRIDAY
25)             6.SATURDAY
26)             7.SUNDAY
27)             """;
28)             }
29)          }
30)       };
31)       System.out.println("You Entered :"+result);
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

```
32)    }
33) }
```

OUTPUT

Enter Day Of WEEK : MONDAY
You Entered :Weekday

OUTPUT

Enter Day Of WEEK : SUNDAY
You Entered :Weekend

OUTPUT

Enter Day Of WEEK : XXXDAY
You Entered :XXXDAY
Value must be in
1.MONDAY
2.TUESDAY
3.WEDNESDAY
4.THURSDAY
5.FRIDAY
6.SATURDAY
7.SUNDAY

Note: To run the above Example we must enable Java14 preview features, for this, we have to use the following changes in IntelliJ Idea

1) **Right Click on Project.**
2) **Open Module Settings**
3) **At Language Level, Select "14(preview) Records, Patterns, Text Blocks" option**
4) **Click on "OK" button.**

2) Pattern Matching for instanceof (Preview)

It was introduced in JAVA12 version as preview feature, it allows instanceof operator with pattern matching capability.

JAVA14 is also making this feature as preview feature, no changes in its feature.

Before Java14 and 12, 'instanceof' operator:

```
1)  package java14features;
2)  public class Test {
3)      public static void main(String[] args) {
4)          Object obj = "Durga Software Solutions";
5)          if(obj instanceof String){
6)              String str = (String)obj;
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

```
7)          System.out.println(str.toUpperCase());
8)        }else{
9)          System.out.println("obj is not having String instance");
10)       }
11)   }
12) }
```

OUTPUT

DURGA SOFTWARE SOLUTIONS

In JAVA12 and JAVA14 version:

```
1)  package java14features;
2)
3)  public class Test {
4)     public static void main(String[] args) {
5)        Object obj = "Durga Software Solutions";
6)        if(obj instanceof String str){
7)          System.out.println(str.toUpperCase());
8)        }else{
9)          System.out.println("obj is not having String instance");
10)       }
11)   }
12) }
```

OUTPUT

DURGA SOFTWARE SOLUTIONS

3) Helpful NullPointerExceptions:

In JAVA14 version, NullpointerException was redesigned with more meaningfull manner.

EX:

```
1)  package java14features;
2)
3)  import java.util.Date;
4)
5)  public class Test {
6)     public static void main(String[] args) {
7)        Date d = null;
8)        System.out.println("To Day : "+d.toString());
9)     }
10) }
```

If we run the above application Before JAVA14 version:

```
Exception in thread "main" java.lang.NullPointerException
        at java14features.Test.main(Test.java:8)
```

In Java14 version, if we want to get NullPointerException helpfull message then we have to execute Java program with "-XX:+ShowCodeDetailsInExceptionMessages" flag along with "java" command.

```
D:\java14features>javac Test.java

D:\java14features>java -XX:+ShowCodeDetailsInExceptionMessages Test
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"java.util.Date.toString()" because "<local1>" is null
    at Test.main(Test.java:7)

D:\java14features>
```

Note: To run the above program and to get NullPointerException Helpfull Message then we have to set "-XX:+ShowCodeDetailsInExceptionMessages" flag in VM Option in in Run/Debug Configurations.

1) **Selct "Run" in Menubar**
2) **Select "Edit Configurations".**
3) **Provide "-XX:+ShowCodeDetailsInExceptionMessages" in Edit Configurations.**
4) **Click on "Apply" then "OK" buttons.**

Run the above program we will get the following output.

```
java.lang.NullPointerException: Cannot invoke "java.util.Date.toString()" because "d" is
null
        at java14features.Test.main(Test.java:9)
```

4) Records (Preview):

In general, in Java applications, we are able to write data carriers like Java bean class, where in Java bean classes we are able to provide the following elements.

1) **Private properties**
2) **Public accessor and mutator methods**
3) **Inplace of mutator methods we may declare parameterized constructors depending on application requirements**
4) **equals(), hashcode() and toString() methods as per the requirement.**

EX:

```
1)  package java14features;
2)
3)  class Employee{
4)     private int eno;
5)     private String ename;
6)     private float esal;
7)     private String eaddr;
8)
9)     Employee(int eno, String ename, float esal, String eaddr){
10)       this.eno = eno;
11)       this.ename = ename;
12)       this.esal = esal;
13)       this.eaddr = eaddr;
14)    }
15)
16)    public int getEno() {
17)       return eno;
18)    }
19)
20)    public String getEname() {
21)       return ename;
22)    }
23)
24)    public float getEsal() {
25)       return esal;
26)    }
27)
28)    public String getEaddr() {
29)       return eaddr;
30)    }
31) }
32) public class Test {
33)    public static void main(String[] args) {
34)       Employee emp = new Employee(111,"AAA",5000,"Hyd");
35)       System.out.println("Employee Details");
36)       System.out.println("----------------------");
37)       System.out.println("Employee Number    : "+emp.getEno());
38)       System.out.println("Employee Name      : "+emp.getEname());
39)       System.out.println("Employee Salary    : "+emp.getEsal());
40)       System.out.println("Employee Address   : "+emp.getEaddr());
41)
42)    }
43) }
```

OUTPUT

```
Employee Details
----------------------
Employee Number    : 111
Employee Name      : AAA
Employee Salary    : 5000.0
Employee Address   : Hyd
```

In the above example, in Employee class we have provided lot of Boilarplatcode like parameterized constructor, accessor methods,.....

To remove the boilarplatcode in the above application JAVA14 version has provided a preview feature that is "Record".

Record is a simple Data carrier, it will include properties, constructor , accessor methods , toString(),
hashCode(), equals(),.....

Syntax:

```
record Name(parameterList){
}
```

EX:

```
record Employee(int eno, String ename, float esal, String eaddr){
}
```

If we compile the above code the compiler will translate the above code like below.

```
final class Employee extends java.lang.Record {
  public Employee(int, java.lang.String, float, java.lang.String);
  public java.lang.String toString();
  public final int hashCode();
  public final boolean equals(java.lang.Object);
  public int eno();
  public java.lang.String ename();
  public float esal();
  public java.lang.String eaddr();
}
```

We are able to get the above code with "javap" command after the compilation.

From the above, we will conclude that,

1. Every record must be converted to a final class
   Note: If record is final class then it is not possible to define inheritance between records.

2. Every record should be a child class to java.lang.Record, where java.lang.Record is providing hashCode(), equals(), toString() methods to record.

```
public abstract class java.lang.Record {
  protected java.lang.Record();
  public abstract boolean equals(java.lang.Object);
  public abstract int hashCode();
  public abstract java.lang.String toString();
}
```

3. Every records contains a parameterized constructor called as Canonical Constructor, where parameters of canonical constructor are same as the parameter types which we provided in record declaration.
4. In record, all the parameters are converted as private and final, we are unable to access them in out side of the record and we are unable to modify their values.
5. In record, for each and every property a seperate accessor method is provided , where accessor methods names are same as the property names like propertyName() , it will not follow getXXX() methods convention.
6. In Records, toString() method was implemented insuch a way to return a string contains "RecordName[Prop1=Value1,Prop2=Value2,....]"
7. In Records, equals() method was implemented in such a way to perform content comparision instead of references comparision.

Note: In JAVA/J2EE applications, we are able to use records as an alternatives to Java Bean classes.

EX:

```
1)  package java14features;
2)  record Employee(int eno, String ename, float esal, String eaddr){
3)
4)  }
5)  public class Test {
6)     public static void main(String[] args) {
7)        Employee emp = new Employee(111,"AAA",5000,"Hyd");
8)        System.out.println("Employee Details");
9)        System.out.println("----------------------");
10)       System.out.println("Employee Number   : "+emp.eno());
11)       System.out.println("Employee Name     : "+emp.ename());
12)       System.out.println("Employee Salary   : "+emp.esal());
13)       System.out.println("Employee Address  : "+emp.eaddr());
14)    }
15) }
```

OUTPUT

```
Employee Details
----------------------
Employee Number    : 111
Employee Name      : AAA
Employee Salary    : 5000.0
Employee Address   : Hyd
```

EX:

```
1)  package java14features;
2)
3)  record Employee(int eno, String ename, float esal, String eaddr){
4)
5)  }
6)  public class Test {
7)     public static void main(String[] args) {
8)        Employee emp1 = new Employee(111,"Durga", 5000, "Hyd");
9)        Employee emp2 = new Employee(111,"Durga", 5000, "Hyd");
10)       Employee emp3 = new Employee(222,"BBB", 6000, "Hyd");
11)       System.out.println(emp1);
12)       System.out.println(emp2);
13)       System.out.println(emp3);
14)       System.out.println(emp1.equals(emp2));
15)       System.out.println(emp1.equals(emp3));
16)
17)    }
18) }
```

Output

```
Employee[eno=111, ename=Durga, esal=5000.0, eaddr=Hyd]
Employee[eno=111, ename=Durga, esal=5000.0, eaddr=Hyd]
Employee[eno=222, ename=BBB, esal=6000.0, eaddr=Hyd]
true
false
```

In record we are able to declare our own variables in the body part, but, it must be static, because, Record body is not allowing instance variables.

EX:

```
1)  package java14features;
2)  record Employee(int eno, String ename, float esal, String eaddr){
3)     static String eemail;
4)
5)     public static String getEemail() {
6)        return eemail;
7)     }
8)
9)     public static void setEemail(String eemail) {
10)       Employee.eemail = eemail;
11)    }
12) }
13) public class Test {
14)    public static void main(String[] args) {
15)       Employee emp = new Employee(111,"AAA",5000,"Hyd");
16)       emp.setEemail("aaa@gmail.com");
17)       System.out.println("Employee Details");
18)       System.out.println("----------------------");
19)       System.out.println("Employee Number    : "+emp.eno());
20)       System.out.println("Employee Name      : "+emp.ename());
21)       System.out.println("Employee Salary    : "+emp.esal());
22)       System.out.println("Employee Address   : "+emp.eaddr());
23)       System.out.println("Employee Email Id  : "+emp.getEemail());
24)
25)    }
26) }
```

OUTPUT

```
Employee Details
----------------------
Employee Number    : 111
Employee Name      : AAA
Employee Salary    : 5000.0
Employee Address   : Hyd
Employee Email Id  : aaa@gmail.com
```

IN Recordfs, we are able to manipulate accessor methods body by writting explicitly.

EX:

```
1)  package java14features;
2)
3)  record User(String fname, String lname, String address){
4)     public String fname(){
5)        return "My First Name : "+fname;
6)     }
7)     public String lname(){
8)        return "My Last Name  : "+lname;
9)     }
10)    public String address(){
11)       return "My Address is : "+address;
12)    }
13) }
14) public class Test {
15)    public static void main(String[] args) {
16)       User user = new User("Java14 version", "JAVA", "Oracle");
17)       System.out.println("User Details");
18)       System.out.println("----------------");
19)       System.out.println(user.fname());
20)       System.out.println(user.lname());
21)       System.out.println(user.address());
22)    }
23) }
```

OUTPUT

```
User Details
----------------
My First Name : Java14 version
My Last Name  : JAVA
My Address is : Oracle
```

In records, if we want to declare our own constructors then it is possible with the following consitions.

1. First we have to implement canonical constructor explicitly.

2. Declare our own constuctor and it must access canonical constructor by using this keyword.

EX:

```
1)  package java14features;
2)  record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)        this(111,"AAA",5000,"Hyd");
5)     }
6)     public Employee(int eno,String ename, float esal){
7)        this(eno,ename,esal,"Hyd");
8)     }
9)     public Employee(int eno, String ename, float esal, String eaddr){
10)       this.eno = eno;
11)       this.ename = ename;
12)       this.esal = esal;
13)       this.eaddr = eaddr;
14)    }
15)
16) }
17) public class Test {
18)    public static void main(String[] args) {
19)       Employee emp = new Employee();
20)       System.out.println("Employee Details");
21)       System.out.println("----------------------");
22)       System.out.println("Employee Number    : "+emp.eno());
23)       System.out.println("Employee Name      : "+emp.ename());
24)       System.out.println("Employee Salary    : "+emp.esal());
25)       System.out.println("Employee Address   : "+emp.eaddr());
26)       System.out.println();
27)       Employee emp1 = new Employee(111,"AAA", 5000);
28)       System.out.println("Employee Details");
29)       System.out.println("----------------------");
30)       System.out.println("Employee Number    : "+emp1.eno());
31)       System.out.println("Employee Name      : "+emp1.ename());
32)       System.out.println("Employee Salary    : "+emp1.esal());
33)       System.out.println("Employee Address   : "+emp1.eaddr());
34)    }
35) }
```

EX:

```
1)  package java14features;
2)  record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)        this(111,"AAA",5000,"Hyd");
5)     }
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

```
6)    public Employee(int eno){
7)       this(eno,"AAA", 5000, "Hyd");
8)    }
9)    public Employee(int eno, String ename){
10)      this(eno, ename, 5000, "Hyd");
11)   }
12)   public Employee(int eno,String ename, float esal){
13)      this(eno,ename,esal,"Hyd");
14)   }
15)   public Employee(int eno, String ename, float esal, String eaddr){
16)      this.eno = eno;
17)      this.ename = ename;
18)      this.esal = esal;
19)      this.eaddr = eaddr;
20)   }
21)   public void getEmpDetails(){
22)      System.out.println("Employee Details");
23)      System.out.println("----------------------");
24)      System.out.println("Employee Number    : "+eno);
25)      System.out.println("Employee Name      : "+ename);
26)      System.out.println("Employee Salary    : "+esal);
27)      System.out.println("Employee Address   : "+eaddr);
28)   }
29)}
30)public class Test {
31)   public static void main(String[] args) {
32)      Employee emp1 = new Employee();
33)      emp1.getEmpDetails();
34)      System.out.println();
35)      Employee emp2 = new Employee(222);
36)      emp1.getEmpDetails();
37)      Employee emp3 = new Employee(333, "BBB");
38)      emp3.getEmpDetails();
39)      System.out.println();
40)      Employee emp4 = new Employee(444,"CCC", 7000);
41)      emp4.getEmpDetails();
42)      Employee emp5 = new Employee(555,"DDD",8000, "Chennai");
43)      emp5.getEmpDetails();
44)   }
45)}
```


DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

OUTPUT

```
Employee Details
----------------------
Employee Number   : 111
Employee Name     : AAA
Employee Salary   : 5000.0
Employee Address  : Hyd

Employee Details
----------------------
Employee Number   : 111
Employee Name     : AAA
Employee Salary   : 5000.0
Employee Address  : Hyd

Employee Details
----------------------
Employee Number   : 333
Employee Name     : BBB
Employee Salary   : 5000.0
Employee Address  : Hyd

Employee Details
----------------------
Employee Number   : 444
Employee Name     : CCC
Employee Salary   : 7000.0
Employee Address  : Hyd

Employee Details
----------------------
Employee Number   : 555
Employee Name     : DDD
Employee Salary   : 8000.0
Employee Address  : Chennai
```

EX:

```
1)  package java14features;
2)  record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)        this(111);
5)     }
6)     public Employee(int eno){
```

```
7)      this(eno,"AAA");
8)    }
9)    public Employee(int eno, String ename){
10)     this(eno, ename, 5000);
11)   }
12)   public Employee(int eno,String ename, float esal){
13)     this(eno,ename,esal,"Hyd");
14)   }
15)   public Employee(int eno, String ename, float esal, String eaddr){
16)     this.eno = eno;
17)     this.ename = ename;
18)     this.esal = esal;
19)     this.eaddr = eaddr;
20)   }
21)   public void getEmpDetails(){
22)     System.out.println("Employee Details");
23)     System.out.println("----------------------");
24)     System.out.println("Employee Number   : "+eno);
25)     System.out.println("Employee Name     : "+ename);
26)     System.out.println("Employee Salary   : "+esal);
27)     System.out.println("Employee Address  : "+eaddr);
28)   }
29) }
30) public class Test {
31)   public static void main(String[] args) {
32)     Employee emp1 = new Employee();
33)     emp1.getEmpDetails();
34)   }
35) }
```

OUTPUT

```
Employee Details
----------------------
Employee Number   : 111
Employee Name     : AAA
Employee Salary   : 5000.0
Employee Address  : Hyd
```

Note: In the above example, we accessed one constructor to another constructor by using 'this()' keyword, so Records are allowing "Constuctor Chaining".

Note: in the above all constructors , we have provided more than one constructor in single record , so that, Records are allowing "Constructor Overloading".

In Record, we are able to declare a constructor with out pareameter and with out () then that constructor is called as "Compact Constructor", it will be accessed just before executing canonical constructor and it will be used mainly for Data validations.

EX:

```
1)  package java14features;
2)  record A(int i, int j){
3)     public A{
4)        System.out.println("Compact Constructor");
5)     }
6)
7)     public void add(){
8)        System.out.println("ADD : "+(i+j));
9)     }
10) }
11) public class Test {
12)    public static void main(String[] args) {
13)       A a = new A(10,20);
14)       a.add();
15)    }
16) }
```

OUTPUT

ADD : 30

EX:

```
1)  package java14features;
2)  record User(String name, String pwd, int age, String email){
3)     public User{
4)        if(name == null || name.equals("")){
5)           System.out.println("Name is Required");
6)        }
7)        if(pwd == null || pwd.equals("")){
8)           System.out.println("Password is Requird");
9)        }else{
10)          if(pwd.length() < 6){
11)             System.out.println("Password length must be minimum 6 characters");
12)          }
13)          if(pwd.length() > 10){
14)             System.out.println("Password length must be maximum 10 characters");
15)          }
16)       }
17)       if(age <= 0){
```

```
18)         System.out.println("Age is required and it must be +ve value");
19)       }else{
20)         if(age < 18 || age > 25){
21)           System.out.println("Age must be in the range from 18 to 25");
22)         }
23)       }
24)       if(email == null || email.equals("")){
25)         System.out.println("Email Id is Required");
26)       }else{
27)         if(!email.endsWith("@durgasoft.com")){
28)           System.out.println("Email Id is Invalid");
29)         }
30)       }
31)    }
32)    public void getUserDetails(){
33)       System.out.println("Uuer Details");
34)       System.out.println("----------------");
35)       System.out.println("User Name     : "+name);
36)       System.out.println("User Password  : "+pwd);
37)       System.out.println("User Age      : "+age);
38)       System.out.println("User Email    : "+email);
39)    }
40) }
41) public class Test {
42)    public static void main(String[] args) {
43)       User user = new User("Durga","durga123",22,"durga@durgasoft.com");
44)       user.getUserDetails();
45)    }
46) }
```

In Java, records are not extending any class,because, records are final classes and it is not possible to provide inheritance relation between records.

EX:
---
record Person(){

}
record Employee() extends Person{

}
Status: Invalid, Compilation Error

IN Java, records can implement interface/Iterfaces.

EX:

```
1)  package java14features;
2)
3)  import java.io.Serializable;
4)
5)  interface Car{
6)     public void getCarDetails();
7)  }
8)  record FordCar(String model, String type, int price) implements Car, Serializable {
9)     @Override
10)    public void getCarDetails() {
11)       System.out.println("Ford car Details");
12)       System.out.println("----------------------");
13)       System.out.println("Car Model   : "+model());
14)       System.out.println("Car Type    : "+type());
15)       System.out.println("Car Price   : "+price());
16)    }
17) }
18) public class Test {
19)    public static void main(String[] args) {
20)       Car fordCar = new FordCar("2015","EchoSport",1200000);
21)       fordCar.getCarDetails();
22)    }
23) }
```

OUTPUT

```
Ford car Details
----------------------
Car Model   : 2015
Car Type    : EchoSport
Car Price   : 1200000
```

In Java14, we are able to get Records details by using Reflection API.
IN Java14, java.lang.Class has provided the following methods.
1. public boolean isRecord()
--> It will check whether the component is Record or not.

2. public RecordComponent[] getRecordComponents()
--> It able to return all parameters of the Records in the form of RecordComponent[].

Note: Where RecordComponent is able to provide Single Record Component metadata.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 88 85 25 26 27, 72 07 21 24 27/28 | www.durgasoftonline.com
Maii: durgasoftonline@gmail.com

EX:

```
1)  package java14features;
2)
3)  import java.lang.reflect.RecordComponent;
4)
5)  record Customer(String cid, String cname, String caddr) {
6)  }
7)  public class Test {
8)     public static void main(String[] args) {
9)        System.out.println("Is Customer Record?   : "+Customer.class.isRecord());
10)       System.out.print("Customer Record Components  : ");
11)       for(RecordComponent comp: Customer.class.getRecordComponents()){
12)          System.out.print(comp.getName()+"  ");
13)       }
14)    }
15) }
```

OUTPUT

```
Is Customer Record?   : true
Customer Record Components  : cid  cname  caddr
```

5) Text Blocks (Second Preview)

Text Blocks are introduced in JAVA13 as Preview version, it allows to declare a string value in more than one line and it is very much usefull when we want tp write html code or JSON code or Sql queries in Java programs.

JAVA14 version made Text Blocks still Preview feature, but, it has allowed to use \[Back Slash] symbol to improve look and feel and it will provide multiple lines of String into single line and \s to preserve trainling spaces in the lines.

EX:

```
1)  package java14features;
2)
3)  import java.lang.reflect.RecordComponent;
4)  public class Test {
5)     public static void main(String[] args) {
6)        String address = """
7)              Durga Software Solutions\
8)              202, HMDA, Mitrivanam\
9)              Ameerpet\
10)             Hyderabad-38\
11)             """;
```

```
12)       System.out.println(address);
13)       System.out.println();
14)    }
15) }
```

OUTPUT

Durga Software Solutions202, HMDA, MitrivanamAmeerpetHyderabad-38

EX:

```
1)  package java14features;
2)
3)  import java.lang.reflect.RecordComponent;
4)  public class Test {
5)     public static void main(String[] args) {
6)        String address = """
7)              Durga Software Solutions\s
8)              202, HMDA, Mitrivanam\s
9)              Ameerpet\s
10)             Hyderabad-38\s
11)             """;
12)       System.out.println(address);
13)       System.out.println();
14)    }
15) }
```

OUTPUT

Durga Software Solutions
202, HMDA, Mitrivanam
Ameerpet
Hyderabad-38

Note: In the above , Every line has Single Space at end.

In Text Block, we can use both \s for space and \ for keeping mnultiple lines in single line.

EX:

```
1)  package java14features;
2)
3)  import java.lang.reflect.RecordComponent;
4)  public class Test {
5)     public static void main(String[] args) {
6)        String address = """
```

```
7)              Durga Software Solutions\s\s\
8)              202, HMDA, Mitrivanam\s\s\
9)              Ameerpet\s\s\
10)             Hyderabad-38\s\s\
11)             """;
12)         System.out.println(address);
13)         System.out.println();
14)
15)     }
16) }
```

OUTPUT

```
Durga Software Solutions  202, HMDA, Mitrivanam  Ameerpet  Hyderabad-38
```