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)cgywbnlk yfdrf

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.
Syntax:

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

EX:

```
package java14features;

import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        var scanner = new Scanner(System.in);
        System.out.print("Enter Day Of WEEK : ");
        var value = scanner.next().toUpperCase();
        var result = switch (value){
            case "MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY" ->
"Weekday";
            case "SATURDAY","SUNDAY" -> "Weekend";
            default ->{
                if(value.isEmpty()){
                    yield "value to switch is empty, please check";
                }else{
                    yield value+"""

                Value must be in
                1.MONDAY
                2.TUESDAY
                3.WEDNESDAY
                4.THURSDAY
                5.FRIDAY
                6.SATURDAY
                7.SUNDAY
                """;
                }
            }
        };
        System.out.println("You Entered :"+result);
    }
}
```

OP:

```
Enter Day Of WEEK : MONDAY
You Entered :Weekday
```

OP:

```
Enter Day Of WEEK : SUNDAY
You Entered :Weekend
```

OP:

```
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:

```
package java14features;
public class Test {
    public static void main(String[] args) {
        Object obj = "Durga Software Solutions";
        if(obj instanceof String){
            String str = (String)obj;
            System.out.println(str.toUpperCase());
        }else{
            System.out.println("obj is not having String instance");
        }
    }
}
```

OP:

```
---
DURGA SOFTWARE SOLUTIONS
```

In JAVA12 and JAVA14 version:

```
package java14features;

public class Test {
    public static void main(String[] args) {
        Object obj = "Durga Software Solutions";
        if(obj instanceof String str){
            System.out.println(str.toUpperCase());
        }else{
            System.out.println("obj is not having String instance");
        }
    }
}
```

OP:

```
----
DURGA SOFTWARE SOLUTIONS
```

3. Helpful NullPointerExceptions:

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

EX:

```
package java14features;

import java.util.Date;

public class Test {
    public static void main(String[] args) {
        Date d = null;
        System.out.println("To Day : "+d.toString());
    }
}
```

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:

```
----
package java14features;

class Employee{
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;

    Employee(int eno, String ename, float esal, String eaddr){
        this.eno = eno;
        this.ename = ename;
        this.esal = esal;
        this.eaddr = eaddr;
    }

    public int getEno() {
        return eno;
    }

    public String getEname() {
        return ename;
    }

    public float getEsal() {
        return esal;
    }

    public String getEaddr() {
        return eaddr;
    }
}
public class Test {
    public static void main(String[] args) {
        Employee emp = new Employee(111,"AAA",5000,"Hyd");
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+emp.getEno());
        System.out.println("Employee Name      : "+emp.getEname());
        System.out.println("Employee Salary    : "+emp.getEsal());
        System.out.println("Employee Address   : "+emp.getEaddr());

    }
}
```

OP:

```
---
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:

```
package java14features;
record Employee(int eno, String ename, float esal, String eaddr){

}
public class Test {
    public static void main(String[] args) {
        Employee emp = new Employee(111,"AAA",5000,"Hyd");
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+emp.eno());
        System.out.println("Employee Name      : "+emp.ename());
        System.out.println("Employee Salary    : "+emp.esal());
        System.out.println("Employee Address   : "+emp.eaddr());

    }
}
```

OP:

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

EX:

```
package java14features;

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

}
public class Test {
    public static void main(String[] args) {
        Employee emp1 = new Employee(111,"Durga", 5000, "Hyd");
        Employee emp2 = new Employee(111,"Durga", 5000, "Hyd");
        Employee emp3 = new Employee(222,"BBB", 6000, "Hyd");
        System.out.println(emp1);
```

```
        System.out.println(emp2);
        System.out.println(emp3);
        System.out.println(emp1.equals(emp2));
        System.out.println(emp1.equals(emp3));

    }
}
```

```
OP
---
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:
---
package java14features;
record Employee(int eno, String ename, float esal, String eaddr){
    static String eemail;

    public static String getEemail() {
        return eemail;
    }

    public static void setEemail(String eemail) {
        Employee.eemail = eemail;
    }
}
public class Test {
    public static void main(String[] args) {
        Employee emp = new Employee(111,"AAA",5000,"Hyd");
        emp.setEemail("aaa@gmail.com");
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+emp.eno());
        System.out.println("Employee Name      : "+emp.ename());
        System.out.println("Employee Salary    : "+emp.esal());
        System.out.println("Employee Address   : "+emp.eaddr());
        System.out.println("Employee Email Id  : "+emp.getEemail());

    }
}
```

```
OP:
---
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:

```
package java14features;

record User(String fname, String lname, String address){
    public String fname(){
        return "My First Name : "+fname;
    }
    public String lname(){
        return "My Last Name  : "+lname;
    }
    public String address(){
        return "My Address is : "+address;
    }
}
public class Test {
    public static void main(String[] args) {
        User user = new User("Java14 version", "JAVA", "Oracle");
        System.out.println("User Details");
        System.out.println("----------------");
        System.out.println(user.fname());
        System.out.println(user.lname());
        System.out.println(user.address());
    }
}
```

OP:

```
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:

```
package java14features;
record Employee(int eno, String ename, float esal, String eaddr){
    public Employee(){
        this(111,"AAA",5000,"Hyd");
    }
    public Employee(int eno,String ename, float esal){
        this(eno,ename,esal,"Hyd");
    }
    public Employee(int eno, String ename, float esal, String eaddr){
        this.eno = eno;
        this.ename = ename;
        this.esal = esal;
```

```
        this.eaddr = eaddr;
    }

}
public class Test {
    public static void main(String[] args) {
        Employee emp = new Employee();
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+emp.eno());
        System.out.println("Employee Name      : "+emp.ename());
        System.out.println("Employee Salary    : "+emp.esal());
        System.out.println("Employee Address   : "+emp.eaddr());
        System.out.println();
        Employee emp1 = new Employee(111,"AAA", 5000);
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+emp1.eno());
        System.out.println("Employee Name      : "+emp1.ename());
        System.out.println("Employee Salary    : "+emp1.esal());
        System.out.println("Employee Address   : "+emp1.eaddr());


    }
}

EX:
---
package java14features;
record Employee(int eno, String ename, float esal, String eaddr){
    public Employee(){
        this(111,"AAA",5000,"Hyd");
    }
    public Employee(int eno){
        this(eno,"AAA", 5000, "Hyd");
    }
    public Employee(int eno, String ename){
        this(eno, ename, 5000, "Hyd");
    }
    public Employee(int eno,String ename, float esal){
        this(eno,ename,esal,"Hyd");
    }
    public Employee(int eno, String ename, float esal, String eaddr){
        this.eno = eno;
        this.ename = ename;
        this.esal = esal;
        this.eaddr = eaddr;
    }
    public void getEmpDetails(){
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+eno);
        System.out.println("Employee Name      : "+ename);
        System.out.println("Employee Salary    : "+esal);
        System.out.println("Employee Address   : "+eaddr);
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Employee emp1 = new Employee();
        emp1.getEmpDetails();
        System.out.println();
        Employee emp2 = new Employee(222);
        emp1.getEmpDetails();
        Employee emp3 = new Employee(333, "BBB");
        emp3.getEmpDetails();
        System.out.println();
        Employee emp4 = new Employee(444,"CCC", 7000);
        emp4.getEmpDetails();
        Employee emp5 = new Employee(555,"DDD",8000, "Chennai");
        emp5.getEmpDetails();



    }
}
OP:
---
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:
----
package java14features;
```

```
record Employee(int eno, String ename, float esal, String eaddr){
    public Employee(){
        this(111);
    }
    public Employee(int eno){
        this(eno,"AAA");
    }
    public Employee(int eno, String ename){
        this(eno, ename, 5000);
    }
    public Employee(int eno,String ename, float esal){
        this(eno,ename,esal,"Hyd");
    }
    public Employee(int eno, String ename, float esal, String eaddr){
        this.eno = eno;
        this.ename = ename;
        this.esal = esal;
        this.eaddr = eaddr;
    }
    public void getEmpDetails(){
        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number    : "+eno);
        System.out.println("Employee Name      : "+ename);
        System.out.println("Employee Salary    : "+esal);
        System.out.println("Employee Address   : "+eaddr);
    }
}
public class Test {
    public static void main(String[] args) {
        Employee emp1 = new Employee();
        emp1.getEmpDetails();
    }
}

OP:
---
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:

```
---
package java14features;
record A(int i, int j){
    public A{
        System.out.println("Compact Constructor");
    }

    public void add(){
        System.out.println("ADD : "+(i+j));
    }
}
public class Test {
    public static void main(String[] args) {
        A a = new A(10,20);
        a.add();
    }
}

OP:
----
ADD : 30

EX:
---
package java14features;
record User(String name, String pwd, int age, String email){
    public User{
        if(name == null || name.equals("")){
            System.out.println("Name is Required");
        }
        if(pwd == null || pwd.equals("")){
            System.out.println("Password is Requird");
        }else{
            if(pwd.length() < 6){
                System.out.println("Password length must be minimum 6
characters");
            }
            if(pwd.length() > 10){
                System.out.println("Password length must be maximum 10
characters");
            }
        }
        if(age <= 0){
            System.out.println("Age is required and it must be +ve
value");
        }else{
            if(age < 18 || age > 25){
                System.out.println("Age must be in the range from 18
to 25");
            }
        }
        if(email == null || email.equals("")){
            System.out.println("Email Id is Required");
        }else{
            if(!email.endsWith("@durgasoft.com")){
                System.out.println("Email Id is Invalid");
            }
```

```
        }
    }
    public void getUserDetails(){
        System.out.println("Uuer Details");
        System.out.println("----------------");
        System.out.println("User Name      : "+name);
        System.out.println("User Password  : "+pwd);
        System.out.println("User Age       : "+age);
        System.out.println("User Email     : "+email);
    }
}
public class Test {
    public static void main(String[] args) {
        User user = new
User("Durga","durga123",22,"durga@durgasoft.com");
        user.getUserDetails();
    }
}
```

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:

```
---
package java14features;

import java.io.Serializable;

interface Car{
    public void getCarDetails();
}
record FordCar(String model, String type, int price) implements Car,
Serializable {
    @Override
    public void getCarDetails() {
        System.out.println("Ford car Details");
        System.out.println("----------------------");
        System.out.println("Car Model   : "+model());
        System.out.println("Car Type    : "+type());
        System.out.println("Car Price   : "+price());
    }
}
public class Test {
    public static void main(String[] args) {
```

```
        Car fordCar = new FordCar("2015","EchoSport",1200000);
        fordCar.getCarDetails();
    }
}
```

OP:

```
---
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.

EX:

```
---
package java14features;

import java.lang.reflect.RecordComponent;

record Customer(String cid, String cname, String caddr){

}
public class Test {
    public static void main(String[] args) {
        System.out.println("Is Customer Record?   :
"+Customer.class.isRecord());
        System.out.print("Customer Record Components  : ");
        for(RecordComponent comp:
Customer.class.getRecordComponents()){
            System.out.print(comp.getName()+"  ");
        }
    }
}
```

OP:

```
----
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:

```
package java14features;

import java.lang.reflect.RecordComponent;
public class Test {
    public static void main(String[] args) {
        String address = """
                Durga Software Solutions\
                202, HMDA, Mitrivanam\
                Ameerpet\
                Hyderabad-38\
                """;
        System.out.println(address);
        System.out.println();

    }
}
```

OP:

```
Durga Software Solutions202, HMDA, MitrivanamAmeerpetHyderabad-38
```

EX:

```
package java14features;

import java.lang.reflect.RecordComponent;
public class Test {
    public static void main(String[] args) {
        String address = """
                Durga Software Solutions\s
                202, HMDA, Mitrivanam\s
                Ameerpet\s
                Hyderabad-38\s
                """;
        System.out.println(address);
        System.out.println();

    }
}
```

OP:

```
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:

```
package java14features;

import java.lang.reflect.RecordComponent;
public class Test {
    public static void main(String[] args) {
        String address = """
                Durga Software Solutions\s\s\
                202, HMDA, Mitrivanam\s\s\
                Ameerpet\s\s\
                Hyderabad-38\s\s\
                """;
        System.out.println(address);
        System.out.println();

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