# JAVA 14 TOP 3 FEATURES

JDK 14 is the open-source reference implementation of version 14 of the Java SE Platform as specified by by JSR 389 in the Java Community Process.

JDK 14 reached General Availability on 17 March 2020. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

You can check the features and Schedule of the jdk 14 by click the : http://openjdk.java.net/projects/jdk/14/

There are many improvement in java14 like every java version.Some of these are the changes that appear to the developer, and some are the infrastructure changes.For example, garbage collector, low level API etc.

As you can see, there are many innovations in java 14 so it takes a lot of time to talk about them all, so I want to talk about 3 features that can be interesting:

- JEP 358 - Helpful NullPointerExceptions
- JEP 361 - Switch Expressions (Standart)
- JEP 368 - Multiline Text Blocks (Second Preview)

/*

What is the preview ? :

**Preview Features**

- Not standart yet
- Ready to try, review and feedback
- May be changed, even removed

**However, preview features comes disabled by default. You can active these using by :

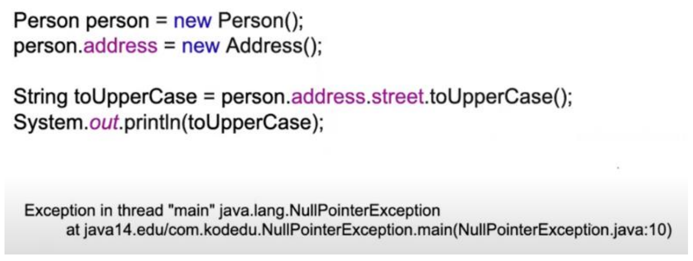
**--enable-preview**

*/

## JEP 358: Helpful NullPointerExceptions

It can be a simple but important feature. So what does it mean?

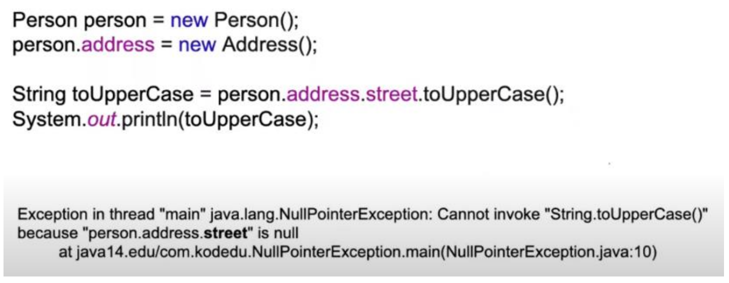
- As seen below when running the code below before JDK 14, we get the NullPointerException error and it doesn't tell us where this error originated.



```
Person person = new Person();
person.address = new Address();

String toUpperCase = person.address.street.toUpperCase();
System.out.println(toUpperCase);
```

```
Exception in thread "main" java.lang.NullPointerException
        at java14.edu/com.kodedu.NullPointerException.main(NullPointerException.java:10)
```

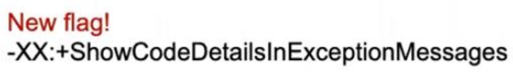
In some cases, it can be quite difficult to find the error. Let's look at the same application with jdk 14.



```
Person person = new Person();
person.address = new Address();

String toUpperCase = person.address.street.toUpperCase();
System.out.println(toUpperCase);
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.toUpperCase()"
because "person.address.street" is null
        at java14.edu/com.kodedu.NullPointerException.main(NullPointerException.java:10)
```

As you can see, it now tells us which method cannot be called and why it cannot be called. So we can immediately change the code and solve the problem. However, when we run jdk14 this feature will not work. This feature is deactivated by default. A new flag has been introduced for this. To activate it, it is necessary to add the following flag as an argument.



New flag!

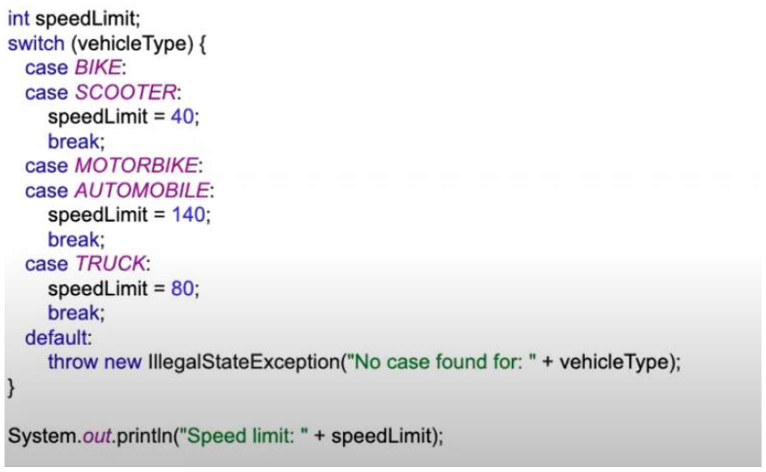
```
-XX:+ShowCodeDetailsInExceptionMessages
```

# JEP 361 - Switch Expressions (Standart)

This feature, which was brought before, is now standardized in jdk 14.

- Pre Switch Expressions

  The switch structure works for each case until it sees a break from top to bottom. break statement used multiple times. So that code complexity occurs.

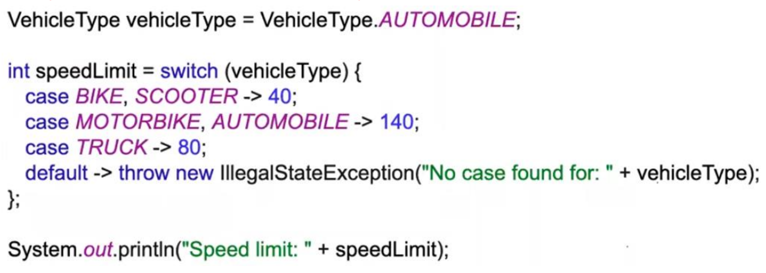


```
int speedLimit;
switch (vehicleType) {
  case BIKE:
  case SCOOTER:
     speedLimit = 40;
     break;
  case MOTORBIKE:
  case AUTOMOBILE:
     speedLimit = 140;
     break;
  case TRUCK:
     speedLimit = 80;
     break;
  default:
     throw new IllegalStateException("No case found for: " + vehicleType);
}

System.out.println("Speed limit: " + speedLimit);
```

- New Switch Expressions

  In the new case, we get rid of both line redundancy and code complexity. Also, as seen below, the switch is used as a method. Again, the case statement is used and if there is more than one case, it is separated by “;” .Furthermore ,a new label came ,the arrow phrase you see. (It is also in lambda expressions).

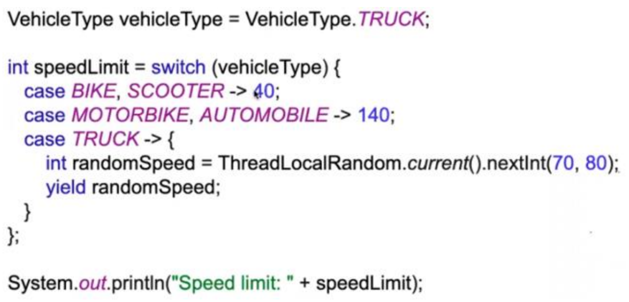
  *Important : default is not required if all enum cases are covered



```
VehicleType vehicleType = VehicleType.AUTOMOBILE;

int speedLimit = switch (vehicleType) {
  case BIKE, SCOOTER -> 40;
  case MOTORBIKE, AUTOMOBILE -> 140;
  case TRUCK -> 80;
  default -> throw new IllegalStateException("No case found for: " + vehicleType);
};

System.out.println("Speed limit: " + speedLimit);
```

- **yield statement**

In some cases, we may have to write logic. Yield statement works like “return” key here and returns that value to case



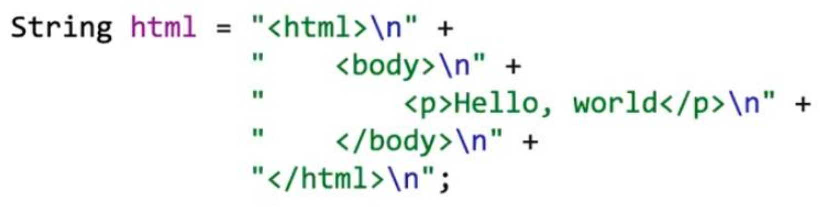
```
VehicleType vehicleType = VehicleType.TRUCK;

int speedLimit = switch (vehicleType) {
  case BIKE, SCOOTER -> 40;
  case MOTORBIKE, AUTOMOBILE -> 140;
  case TRUCK -> {
     int randomSpeed = ThreadLocalRandom.current().nextInt(70, 80);
     yield randomSpeed;
  }
};

System.out.println("Speed limit: " + speedLimit);
```

# JEP 368 - Multiline Text Blocks (Second Preview)

It allows us to write strings, texts and especially multi-line text more easily. It also shows single-line but very long texts as multiline and changes the representation, providing a cleaner appearance.

- **Pre Text Blocks**

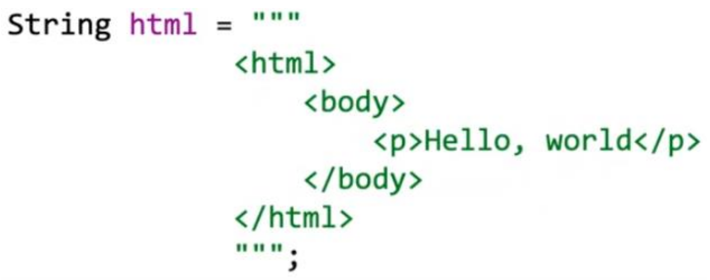
As you can see, we combine multiple lines with “+”. This creates an ugly code.



```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

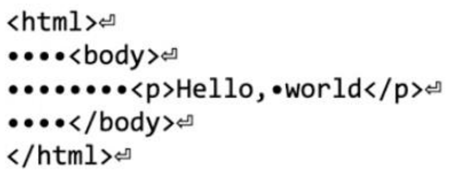
- **Text Blocks**

The structure is created with 3 quotes like below and text is written without using “\n” or “+” expressions.



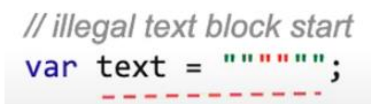
```
String html = """
              <html>
                  <body>
                      <p>Hello, world</p>
                  </body>
              </html>
              """;
```

- Suppose we write a function that replaces the space key with the dot and “\n” expression with the enter icon and that function prints the text to console.Print is below:



```
<html>⏎
••••<body>⏎
••••••••<p>Hello,•world</p>⏎
••••</body>⏎
</html>⏎
```

- In order to define text blocks legally, it is necessary to move to the bottom line after the first 3 quotes opened. Otherwise we get an error:



```
// illegal text block start
var text = """""";
```

Note: The reason why these three changes I told are the best for me is that there have been changes over the conventional structures that we have been using for years.Moreover, these changes very close to developers.