Java 14

Links :-

1. http://openjdk.java.net/projects/jdk/14/
2. https://cr.openjdk.java.net/~iris/se/14/latestSpec/apidiffs/overview-summary.html (API differences)
3. https://www.oracle.com/java/technologies/javase/14-relnote-issues.html#NewFeature

Helpful NullpointerException:-

1. NullpointerException is a million dollar mistake that is done by Java Devs.
2. We get some information in stack trace in logs about the NullpointerException whenever it occurs. This information consists of Class name, method name and line number.

```
Exception in thread "main" java.lang.NullPointerException
    at NPE.main(NPE.java:7)
```

3. But sometime it is not very much helpful. For ex:-

```
    String cityName = employe.address.city.getName();
    a.b = c.d
```

4. In above example, we can see multilevel association and developer will not be able to understand that who is the culprit for Null pointer here.
5. But Java-14 has its solution by providing more details of it. To do so we have to enable a switch.

```
    -XX:+ShowCodeDetailsInExceptionMessages
```

6. Now lets have an example:-

```
public class NPE {
    public static void main(String[] args) {
        String s = null;
        int length = s.length();
        System.out.println(length);
    }
}


O/P:-
```

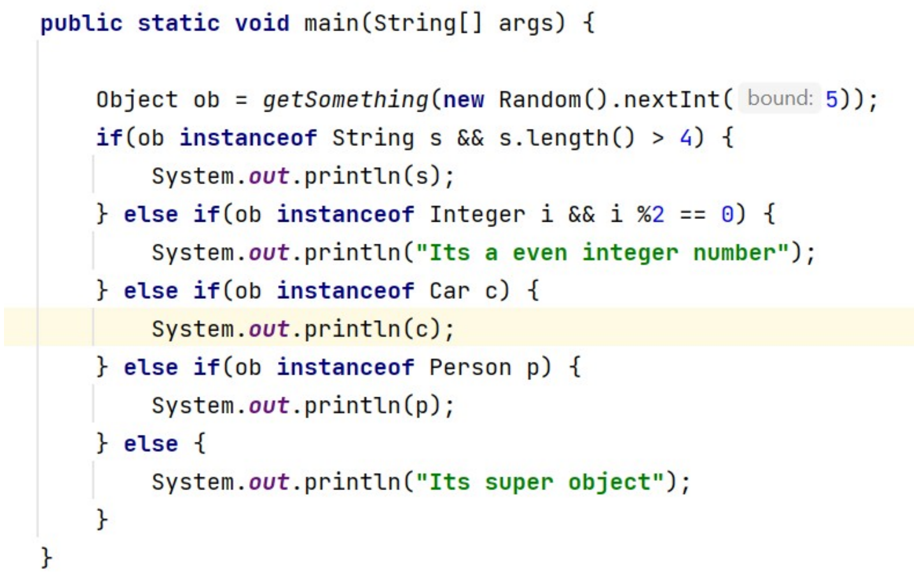
```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()"
because "s" is null
        at java14.npe.NPE.main(NPE.java:7)
```

InstanceOf pattern check:-

1. InstanceOf operator is used to check if an object is of particular type or not. For ex:-

```
    Object ob = "Java 14 features"
    if(ob instanceof String) {
        String str = (String)ob;
    }
```

2. Here code inside the if statement is redundant code. So here is an improvement is done and we can declare variable within the if statement. For ex:



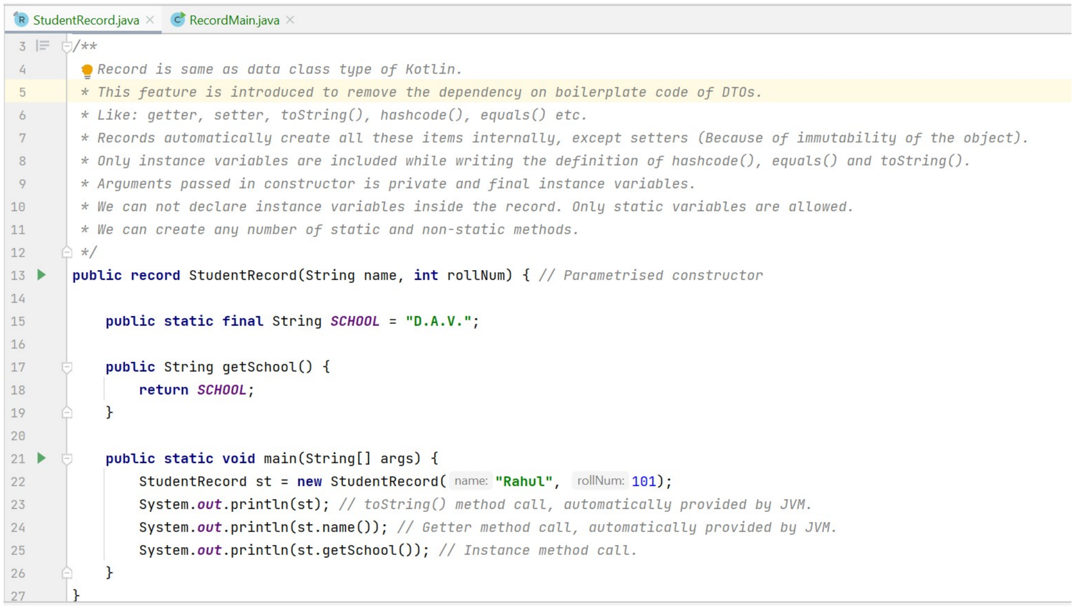
```
public static void main(String[] args) {

    Object ob = getSomething(new Random().nextInt( bound: 5));
    if(ob instanceof String s && s.length() > 4) {
        System.out.println(s);
    } else if(ob instanceof Integer i && i %2 == 0) {
        System.out.println("Its a even integer number");
    } else if(ob instanceof Car c) {
        System.out.println(c);
    } else if(ob instanceof Person p) {
        System.out.println(p);
    } else {
        System.out.println("Its super object");
    }
}
```

3. Legacy code is also supported. But we can save efforts of down casting by using this improvement.

Record:-

1. Record is same as data class type of Kotlin.
2. This feature is introduced to remove the dependency on boilerplate code of DTOs. Like: getter, setter, toString(), hashcode(), equals() etc.

3. Records automatically create all these items internally, except setters (Because of immutability of the object).
4. Only instance variables are included while writing the definition of hashcode(), equals() and toString().
5. Arguments passed in constructor is private and final instance variables.
6. We can not declare instance variables inside the record. Only static variables are allowed.
7. We can create any number of static and non-static methods.
8. Records are implicitly final and cannot be declared as abstract.
9. Records can not be extended further and we can not extend any other class, record in it because by default it is child of java.lang.Record class.
10. We can implement N numbers of interfaces.
11. Example:-



```java
/**
 * Record is same as data class type of Kotlin.
 * This feature is introduced to remove the dependency on boilerplate code of DTOs.
 * Like: getter, setter, toString(), hashcode(), equals() etc.
 * Records automatically create all these items internally, except setters (Because of immutability of the object).
 * Only instance variables are included while writing the definition of hashcode(), equals() and toString().
 * Arguments passed in constructor is private and final instance variables.
 * We can not declare instance variables inside the record. Only static variables are allowed.
 * We can create any number of static and non-static methods.
 */
public record StudentRecord(String name, int rollNum) { // Parametrised constructor

    public static final String SCHOOL = "D.A.V.";

    public String getSchool() {
        return SCHOOL;
    }

    public static void main(String[] args) {
        StudentRecord st = new StudentRecord( name: "Rahul",  rollNum: 101);
        System.out.println(st); // toString() method call, automatically provided by JVM.
        System.out.println(st.name()); // Getter method call, automatically provided by JVM.
        System.out.println(st.getSchool()); // Instance method call.
    }
}
```

12.