JAVA

Version 1.0.0
Niveau requis : 4/7

Evolutions de Java 17 (depuis Java 11)

Sommaire

I. Préambule

I.I. *Objet*

L'objet de ce document est de décrire les évolutions majeures qui ont succéder depuis java 11 pour arriver à la version LTS de JAVA 17.

I.II. *Pré-requis*

Nous allons exploiter dans ce document un projet Java maven 17 sous IntelliJ-Community sous Windows.

Il est nécessaire préalablement d'avoir donc :

- Java 17 : https://bell-sw.com/pages/downloads/
- Maven Apache : https://maven.apache.org/download.cgi
- IntelliJ-Community : https://www.jetbrains.com/idea/download/?section=windows

I.III. *Versions du document*

Version	Date	Auteur	Description
1.0	10/08/2023	Péquignat.eu	Initialisation du document

I.IV. *Documents de référence*

#	Document	Version	Auteur(s)
[R1	https://mydeveloperplanet.com/2021/09/28/whats-new-between-java-11-and-java-17/	September 28, 2021	mydeveloperplanet
[R2	https://bell-sw.com/roadmap/		bell-sw.com
[R3	https://medium.com/@javatechie/the-evolution-of-switch-statement-from-java-7-to-java-17-4b5eee8d29b7	24/12/2021	Java Techie

II. Introduction

II.I. *Java 17 LTS*

II.I.1 Long Term Support

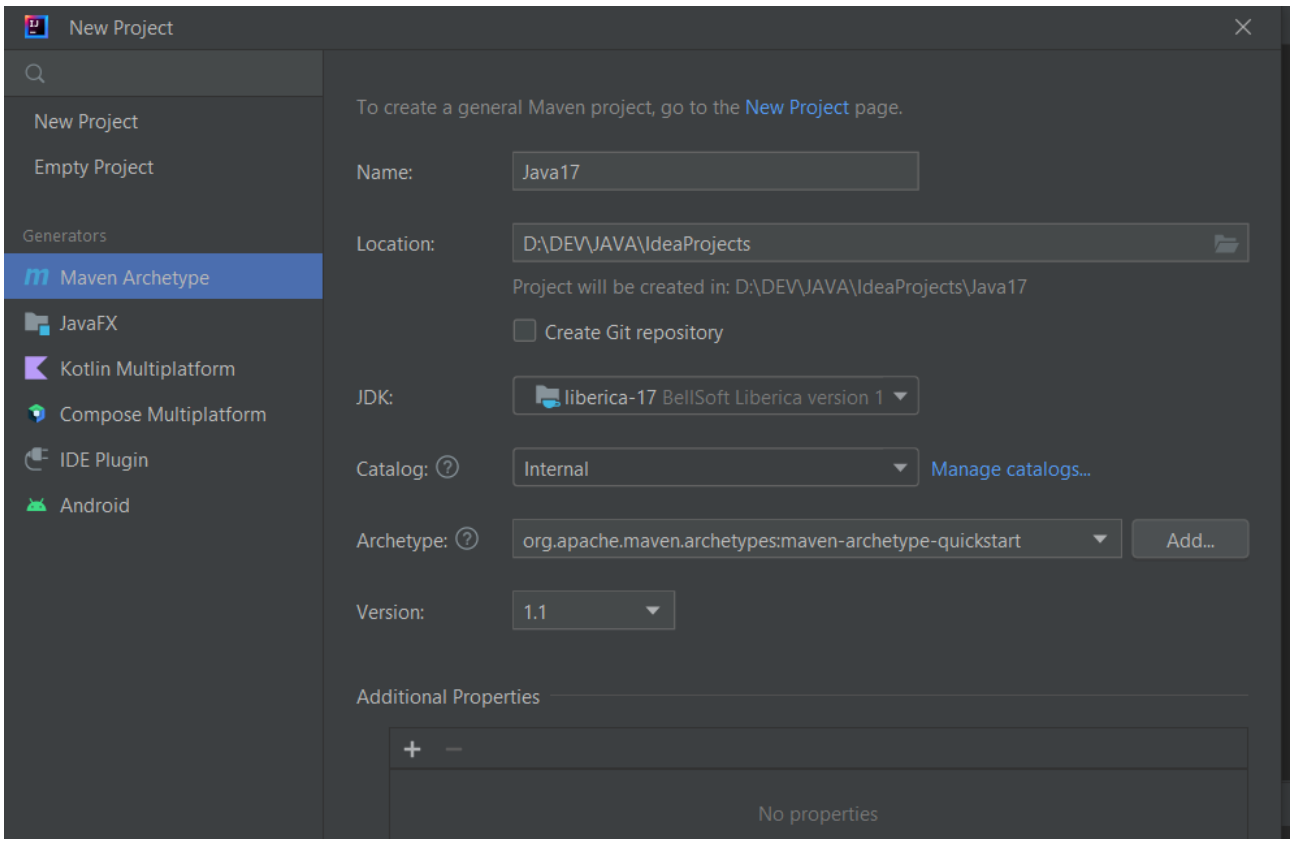
Java 17 est une version de support de longue durée comme pour Java 11.

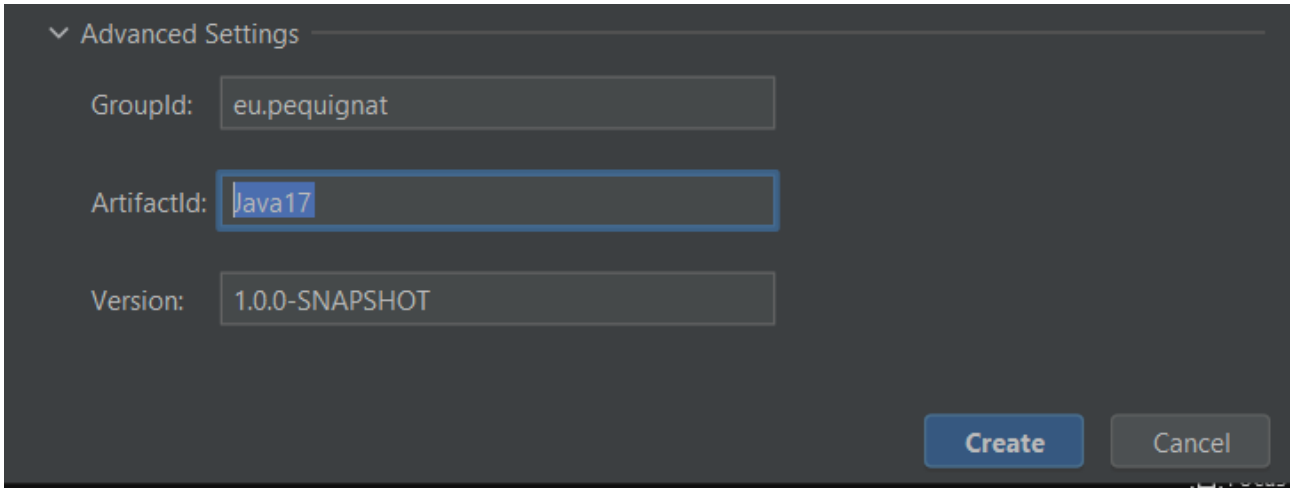
Java 11 a son support jusqu'en septembre 2023, tandis que le support de Java 17 va durer jusqu'en septembre 2026 pour Oracle pour le Support Premier, et septembre 2029 pour le support étendu.
Pour la version 17 de Liberica le support étendu est jusqu'en Mars 2030.

III. Parcours des Evolutions par l'exemple

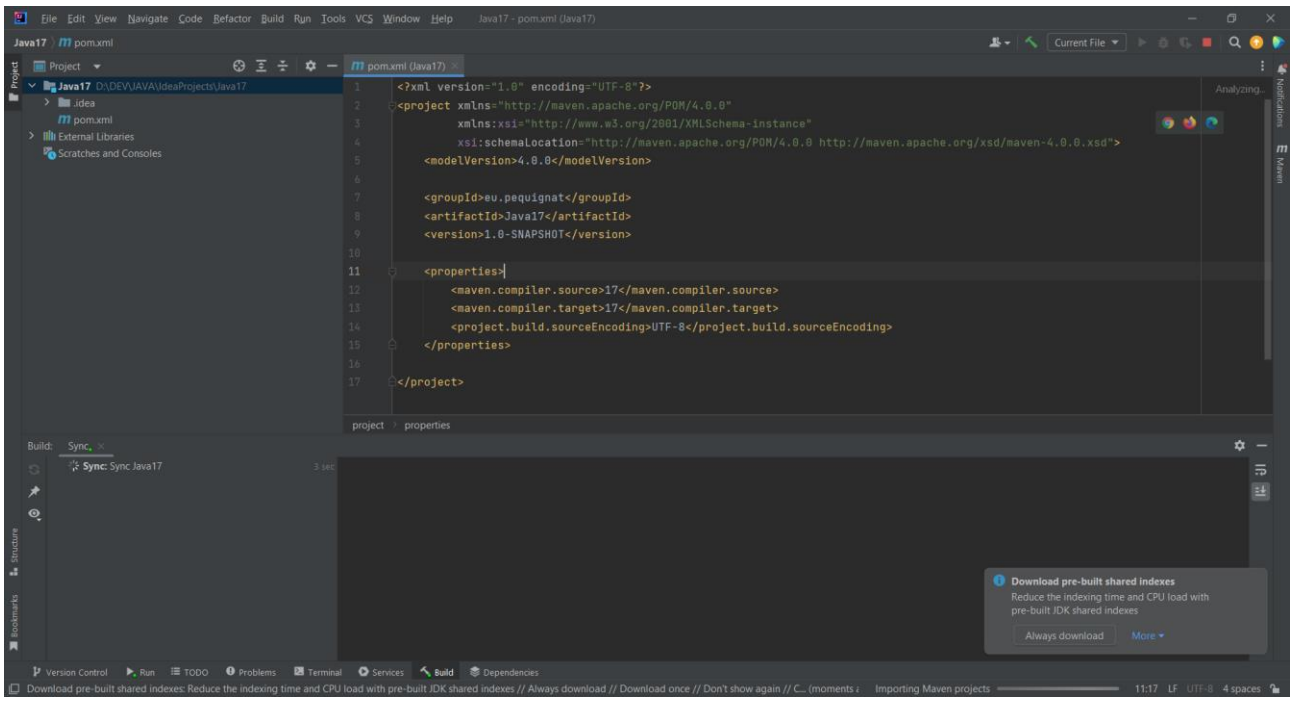
III.I. *Initialisation d'un projet Java 17 sous IntelliJ et Maven*

1. Lancer Intellij et cliquer sur « New Project »
2. Choisir sur la gauche « Maven Archetype »
3. Dans le Champ « Name » du projet : « Java17 »
4. La localisation à votre guise
5. JDK : cliquer sur « Add JDK… »
    a. Rechercher : « C:\Program Files\BellSoft\LibericaJDK-17 »
6. Dans Archtetype : « org.apache.maven.archetypes:maven-archetype-quickstart»
7. Dans « Advanced Settings »
    a. GroupId : eu.pequignat
    b. ArtefactId : Java17
    c. Version : 1.0.0-SNAPSHOT





Cliquer sur « Create »



Le projet s'ouvre, attendre la fin de l'indexation de JDK17 et la fin de la configuration du projet.

III.II. *Projet vide Java17*

III.II.1 Suppression de la classe de Test

La classe de tests générée est inutile.

La supprimer.

III.II.2 Mise du pom.xml

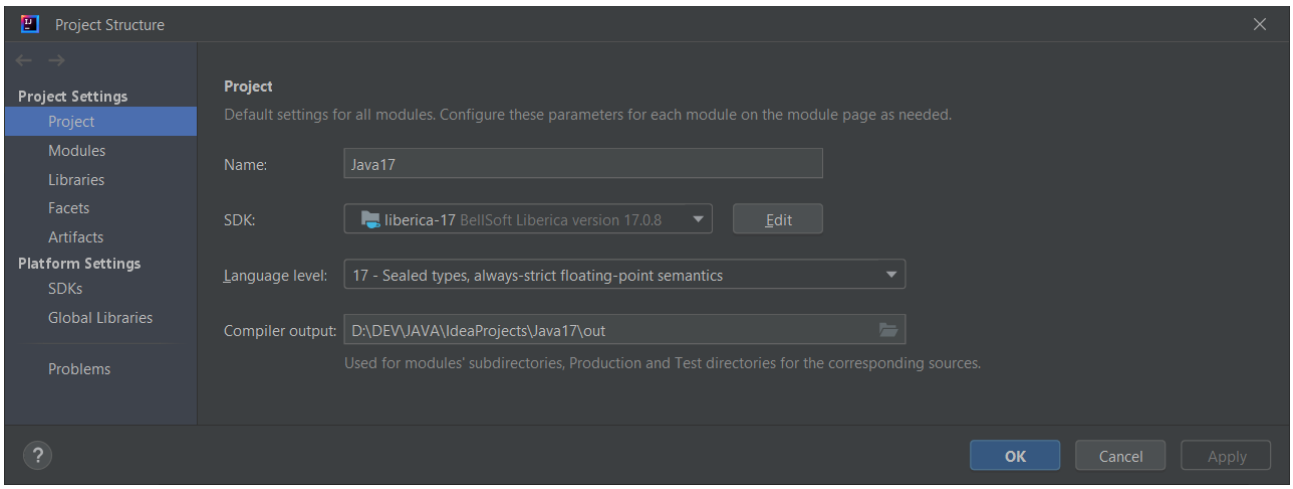
Nous allons personnaliser le fichier pom.xml pour y configurer java en version 17.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>eu.pequignat</groupId>
  <artifactId>Java17</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Java17</name>
  <url>http://maven.apache.org</url>
```

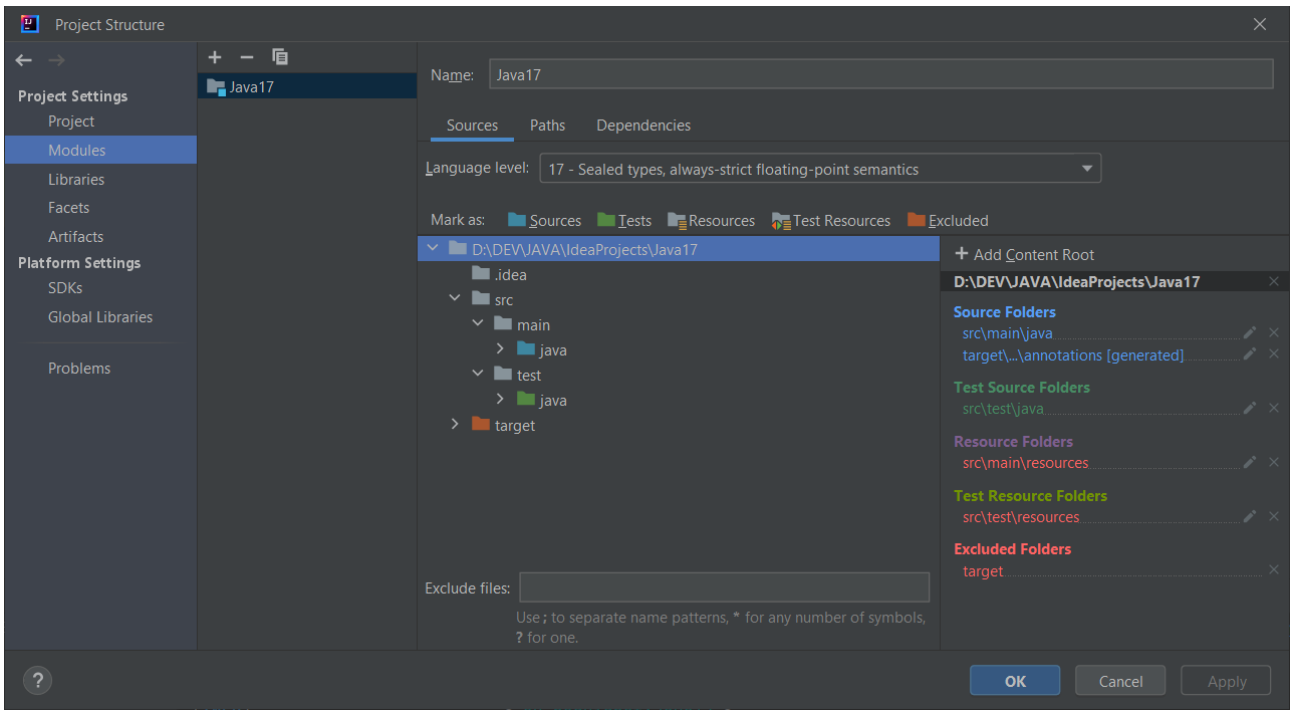
```
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.10.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.6.0</version>
        <configuration>
          <release>17</release>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

III.II.3 Mise de l'écriture dans IntelliJ du module compatible avec Java 17

Dans File > Project Structures > Project :

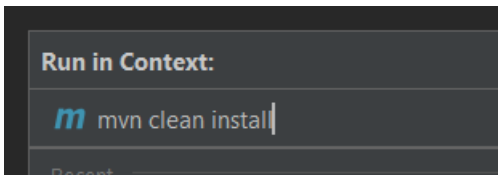


Dans la partie Modules :



III.II.4 Lancer un mvn clean install

Dans la partie maven en haut à droite, lancer un mvn clean install



III.III. *Nouvelle fonctionnalité : Les Text Block*

III.III.1 Java 11

```
package eu.pequignat.textblock;

public class Java11Text {

    public static String getText(){
        String text = "{\n" +
                "   \"name\": \"John Doe\",\n" +
                "   \"age\": 45,\n" +
                "   \"address\": \"Doe Street, 23, Java Town\"\n" +
                "}";
        return text.trim().replaceAll("\\s+", " ").replaceAll("\\r?\\n",
System.lineSeparator());
    }
```

```
}
```

III.III.2 Java 17

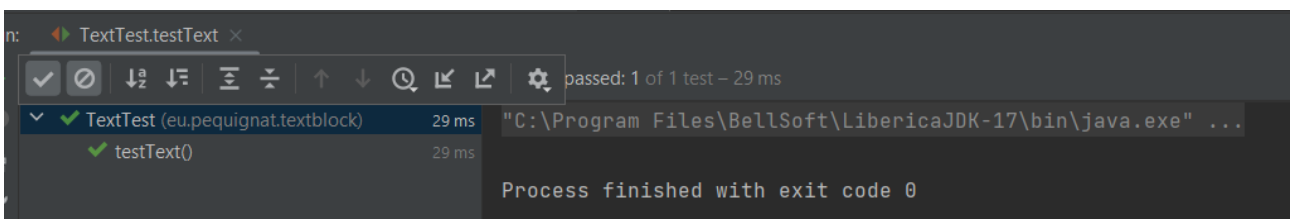
Il est maintenant possible d'écrire à la place de la concaténation avec l'opérateur + ou concat, directement le texte entouré de « """ ».
Cela donne pour le même code rendu :

```
package eu.pequignat.textblock;

public class Java17Text {

    public static String getText(){
        String text = """
{
  "name": "John Doe",
  "age": 45,
  "address": "Doe Street, 23, Java Town"
}
""";
        return text.trim().replaceAll("\\s+", " ").replaceAll("\\r?\\n",
System.lineSeparator());
    }


}
```

III.III.3 Test Unit ?

Vérifions si l'on a bien la même chaine de caractère ?

```
package eu.pequignat.textblock;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TextTest {

    @Test
    public void testText(){
        String textJava11=Java11Text.getText();
        String textJava17=Java17Text.getText();
        Assertions.assertEquals(textJava11,textJava17);
    }

}
```



III.IV. *Nouvelle fonctionnalité : Les Switch*

III.IV.1 Java 11

En java 11, cela conserve la même syntaxe que pour Java 8.

```
package eu.pequignat.switchexpr;

public class Java11Switch {

    public enum Days {
        MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;
    }

    public static String getSwitchClassicWithoutBreak(){
        StringBuilder result = new StringBuilder();
        int i=1;
        switch (i){
            case 1 :
                result.append("One:");
            default:
                result.append("default");
        }
        return result.toString();
    }

    public static String getSwitchClassicWithBreak(){
        StringBuilder result = new StringBuilder();
        int i=1;
        switch (i){
            case 1 :
                result.append("One");
                break;
            default:
                result.append(i);
        }
        return result.toString();
    }

    public static String getSwitchStringClassicWithBreak(){
        StringBuilder result = new StringBuilder();
        final String input="One";
        switch (input){
            case "One" :
                result.append(input);
                break;
            default:
                result.append("default");
        }
        return result.toString();
    }

    public static String getWeekOfDay(){
        Days days = Days.SUNDAY;
        StringBuilder result = new StringBuilder();
        switch (days) {
            case MONDAY:
                result.append("Weekdays");
```

```
                break;
            case TUESDAY:
                result.append("Weekdays");
                break;
            case WEDNESDAY:
                result.append("Weekdays");
                break;
            case THURSDAY:
                result.append("Weekdays");
                break;
            case FRIDAY:
                result.append("Weekdays");
                break;
            case SATURDAY:
                result.append("Weekends");
                break;
            case SUNDAY:
                result.append("Weekends");
                break;
            default:
                result.append("Unknown");
        }
        return result.toString();
    }
}
```

III.IV.2 Java 17 – Nouvelles syntaxes

```
package eu.pequignat.switchexpr;

public class Java17Switch {

    public enum Days {
        MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;
    }

    /*
    THE NEW FUNCTIONS / SYNTAXES
     */

    public static String getNewWeekOfDaySyntaxYield(){
        Days days =  Days.SUNDAY;
        return switch (days) {
            case MONDAY:
                yield "Weekdays";
            case TUESDAY:
                yield "Weekdays";
            case WEDNESDAY:
                yield "Weekdays";
            case THURSDAY:
                yield "Weekdays";
            case FRIDAY:
                yield "Weekdays";
            case SATURDAY:
                yield "Weekends";
            case SUNDAY:
```

```
                yield "Weekends";
            default:
                yield "Unknown";
        };
    }

    public static String getNewWeekOfDaySyntaxLambda(){
        Days days =  Days.SUNDAY;
        return switch (days) {
            case MONDAY ->"Weekdays";
            case TUESDAY -> "Weekdays";
            case WEDNESDAY -> "Weekdays";
            case THURSDAY -> "Weekdays";
            case FRIDAY -> "Weekdays";
            case SATURDAY -> "Weekends";
            case SUNDAY -> "Weekends";
            default -> "Unknown";
        };
    }

    public static String getNewWeekOfDaySyntaxLambdaMultiCase(){
        Days days =  Days.SUNDAY;
        return switch (days) {
            case MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> "Weekdays";
            case SATURDAY, SUNDAY -> "Weekends";
            default -> "Unknown";
        };
    }

}
```

Ici nous pouvons voir l'apparition du mot **yield** ainsi que l'utilisation de lambda et case multiple.

III.IV.3 Test Unit ?

```
package eu.pequignat.switchexpr;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class SwitchTest {

    @Test
    public void testSwitchesJava11WithoutBreak(){
        String switchStrJava11ClassicWOBreak =
Java11Switch.getSwitchClassicWithoutBreak();
        Assertions.assertEquals("One:default", switchStrJava11ClassicWOBreak);
    }

    @Test
    public void testSwitchesJava11WithBreak(){
        String switchStrJava11ClassicWithBreak =
Java11Switch.getSwitchClassicWithBreak();
        Assertions.assertEquals("One", switchStrJava11ClassicWithBreak);
    }
```

```
    @Test
    public void testStringSwitchJava11WithBreak(){
        Assertions.assertEquals("One",
Java11Switch.getSwitchStringClassicWithBreak());
    }

    @Test
    public void testEnumSwitchJava11WithBreak(){
        Assertions.assertEquals("Weekends", Java11Switch.getWeekOfDay());
    }

    @Test
    public void testGetNewWeekOfDaySyntaxYield(){
        Assertions.assertEquals("Weekends",
Java17Switch.getNewWeekOfDaySyntaxYield());
    }

    @Test
    public void testGetNewWeekOfDaySyntaxLambda(){
        Assertions.assertEquals("Weekends",
Java17Switch.getNewWeekOfDaySyntaxLambda());
    }

    @Test
    public void testGetNewWeekOfDaySyntaxLambdaMultiCase(){
        Assertions.assertEquals("Weekends",
Java17Switch.getNewWeekOfDaySyntaxLambdaMultiCase());
    }
}
```

III.V. *Nouvelle interprétation de compilation : instanceof*

Le mot clef instanceof renvoie maintenant une Expression et n'est plus un boolean natif. Il est maintenant possible d'initialiser une variable directement contenant le cast de l'objet dans sa classe vérifiée :

```
package eu.pequignat.instance;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class InstanceOfTest {

    @Test
    public void testInstanceOf(){
        Person p = new Customer();
        if (p instanceof Customer customer){
            Assertions.assertTrue(customer.pay());
        }else{
            Assertions.fail("p instance is not a Customer");
        }
    }


}
```

avec la classe Person et Customer :

```
package eu.pequignat.instance;

public abstract class Person {

}
```

```
package eu.pequignat.instance;

public class Customer extends Person{

    public boolean pay() {
        //Do something
        return true;
    }
}
```

III.VI. *Nouvelle fonctionnalité : Classe Selead*

III.VI.1 Interface ou Classe Abstraite Sealed

La classe abstraite ou interface sealed permet de définir à la compilation les classes autorisées à étendre ou implémenter la classe sealed.

Voyons par l'exemple :

Nous avons défini une interface « IFruitSealed » qui sera public et sealed.

On doit donc définir les classes explicitement qui seront autorisées à implémenter cette interface, Soit :

- AppleFinalSealed : une classe dite « **final »** qui sera donc une feuille
- PearSealed : une classe dite aussi « sealed » qui devra donc elle-même avoir une sous classe déclarée « SubPearFinalSealed »
- AppleNotSealed : une classe dite « non-sealed » qui permet d'avoir autant de sous-type et ainsi pouvoir étendre à souhait avec des sous classes classiques, exemple « OtherInheritanceFruit »

III.VI.2 Codes

```
package eu.pequignat.sealed;

public sealed interface IFruitSealed permits AppleFinalSealed, AppleNotSealed,
PearSealed {
}
```

```java
package eu.pequignat.sealed;

public final class AppleFinalSealed implements IFruitSealed{
}
```

```java
package eu.pequignat.sealed;

public non-sealed class AppleNotSealed implements IFruitSealed{
}
```

```java
package eu.pequignat.sealed;

public class OtherInheritanceFruit extends AppleNotSealed{
}
```

```java
package eu.pequignat.sealed;

public sealed class PearSealed implements IFruitSealed permits
SubPearFinalSealed {
}
```

```java
package eu.pequignat.sealed;

public final class SubPearFinalSealed extends PearSealed{
}
```

III.VII. *Amélioration : NullPointerException*

La gestion de l'exception NullPointerException donne plus de détails sur la cause de cette exception.

```
java.lang.NullPointerException: Cannot invoke "java.lang.Integer.intValue()"
because the return value of "java.util.Map.get(Object)" is null

      at
eu.pequignat.npe.NullPointerExceptionClass.npeMethod(NullPointerExceptionClass.j
ava:13)

      at
eu.pequignat.npe.NullPointerExceptionClassTest.testNEP(NullPointerExceptionClass
Test.java:11)
```

```java
package eu.pequignat.npe;

import java.util.HashMap;
import java.util.Map;

public class NullPointerExceptionClass {

    public static void npeMethod() throws NullPointerException{
        Map<String, Integer> map = new HashMap<>();
        map.put("1", Integer.valueOf(1));
        map.put("2", Integer.valueOf(2));
        map.put("null", null);
        int intValue = ((Integer) map.get("null")).intValue();
    }

}
```

III.VIII. *Nouvelle fonctionnalité : Records*

Les « record » permettent de décrire des POJO en limitant l'écriture à la simple déclaration sur une ligne :

```java
public record GrapeRecord (Color color, int nbrOfPits)  {}
```

Automatiquement les accesseurs sont instanciés automatique. Vous remarquerez que vous ne disposer pas de setter car cela génère une classe immutable, donc non modifiable.

Les méthodes, equals, toString et hashcode sont aussi implémentées automatiquement.

Il est possible de définir du code exécuter à la construction de l'objet comme moyen de validation :

```
package eu.pequignat.record;

import java.awt.*;

public record GrapeRecord (Color color, int nbrOfPits)  {

    public GrapeRecord {
        System.out.println("Parameter color=" + color + ", Field color=" +
this.color());
        System.out.println("Parameter nbrOfPits=" + nbrOfPits + ", Field
nbrOfPits=" + this.nbrOfPits());
        if (color == null) {
            throw new IllegalArgumentException("Color may not be null");
        }
    }

}
```

Voici la classe de Tests Unitaires :

```
package eu.pequignat.record;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.awt.*;

public class GrapeRecordTest {


    @Test
    public void testRecords(){
        GrapeRecord grape1 = new GrapeRecord(Color.BLUE, 1);
        GrapeRecord grape1Copy = new GrapeRecord(grape1.color(),
grape1.nbrOfPits());
        GrapeRecord grape2 = new GrapeRecord(Color.WHITE, 2);
        Assertions.assertNotSame(grape1, grape2);
        System.out.println("Grape 1 is " + grape1);
        System.out.println("Grape grape1 Copy is " + grape1Copy);
        Assertions.assertEquals(grape1Copy, grape1);

        System.out.println("Grape 2 is " + grape2);
        System.out.println("Grape 1 equals grape 2? " + grape1.equals(grape2));
        Assertions.assertNotEquals(grape1, grape2);
    }

    @Test
    public void testRecordsIllegalArgument(){
        try {
            GrapeRecord grape1 = new GrapeRecord(null, 1);
        }catch (Exception e){
            Assertions.assertInstanceOf(IllegalArgumentException.class, e);
        }

    }
}
```

III.IX. *Amélioration syntaxe : Stream.toList()*

Sur la récupération dans une List immutable les objets présents dans un Stream, il est possible de simplifier maintenant en utilisant l'écriture :

```
package eu.pequignat.stream;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java17StreamList {
    public static List<Integer> toList(){
        Stream<Integer> stringStream = Stream.of(1,2,3);
        return stringStream.toList();
    }
}
```

Au lieu de devoir mettre :

```
package eu.pequignat.stream;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java11StreamList {

    public static List<Integer> toList(){
        Stream<Integer> stringStream = Stream.of(1,2,3);
        return stringStream.collect(Collectors.toList());
    }
}
```

Le Test Unitaire :

```
package eu.pequignat.stream;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class StreamListTest {

    @Test
    public void testEqualsList(){

Assertions.assertEquals(Java11StreamList.toList(),Java17StreamList.toList());
    }

}
```

IV.Fin du document