DJILLALI LIABES UNIVERSITY OF SIDI BEL ABBES
FACULTY OF EXACT SCIENCES
DEPARTMENT OF COMPUTER SCIENCES

Module : Algorithmique et Complexité
1ST YEAR OF MASTER'S DEGEREE IN
NETWORKS,INFORMATION SYSTEMS & SECURITY(RSSI)
2021/2022

Comparaison entre algorithmes récursifs pour le calcul de la suite de Fibonacci

Author:
HADJAZI M.Hisham
AMUER Wassim Malik
Group: 01/RSSI

Supervisor:
Dr. MME. BELKHODJA
ZENAIDI Lamia

A paper submitted in fulfillment of the requirements for the
TP-02

November 4, 2021

Contents

Chapter 1

Solutions of Fiche TP-01

Notes regarding this solution :
This solution and the executions of the code in it was done in the following machine :

- *Machine*: Lenovo Ideapad S210
- *CPU*: Intel Celeron 1037U 1800 MHz
- *RAM*: 8GB DDR3l
- *OS* : Linux Mint 20.2 Cinnamon Kernel v.5.4.0-88
- *IDE* : Eclipse IDE for Java Developers Version: 2019-12 (4.14.0)
- *Java version*: 11.0.11

On s'intéresse au calcul des nombres de la suite de Fibonacci avec une **approche récursive**. On rappelle **la formule** utilisée pour le TP1.

F(0)= 0 ; F(1) = 1
F(n) = F (n-1) + F (n-2) pour n > 1

1.1 Question 1 : Compléter la classe Fibo2, ci-jointe, avec 3 méthodes récursives et exécuter le programme pour différentes valeurs de n=0,1,.., 10, ..., 80,...100,...

1.1.1 1. La méthode4 utilise directement la formule avec 2 appels récursifs (récursivité non terminale et une solution naïve)

```
public static BigInteger methode4(int n) {

 BigInteger result = BigInteger.ZERO;

 if (n == 0)
        return BigInteger.ZERO;
 if (n == 1)
        return BigInteger.ONE;

 if (arrayCache[(int) n] != null) {
        return arrayCache[(int) n];
        } else {
        result = methode4(n - 1).add(methode4(n - 2));
        arrayCache[(int) n] = result;
        return result;
 }
}

private static final BigInteger[]
        arrayCache = new BigInteger[100000];

        static {
                arrayCache[0] = BigInteger.ONE;
                arrayCache[1] = BigInteger.ONE;
}
```

1.1.2 2. La méthode5 utilise la formule avec un seul appel récursif et une récursivité terminale (aucune instruction n'est autorisée après l'appel récursif).

```
static BigInteger methode5(
                int n, BigInteger f0, BigInteger f1) {
 if (n == 0)
 return BigInteger.ZERO;
 if (n == 1 || n == 2) {
 return f0;
 }
 return methode5(n - 1, f0.add(f1), f0);
}
```

1.1.3 3. La méthode6 utilise un seul appel récursif et une approche matricielle.

```
private static BigInteger[] methode6(
                BigInteger[] matrix1,
                BigInteger[] matrix2,
                int n) {
if (n == 0)
        return result;

if (n % 2 != 0) {
        return methode6(matrixMultiply(matrix1, matrix1),
                         matrixMultiply(matrix2, matrix1),
                        n / 2);
} else {
        return methode6(matrixMultiply(matrix1, matrix1),
                         matrix2, n / 2);
        }
        }

// Multiplies 2 matrices.
private static BigInteger[] matrixMultiply(BigInteger[] x,
                         BigInteger[] y){

return new BigInteger[] {
        multiply(x[0], y[0]).add(multiply(x[1], y[2])),
        multiply(x[0], y[1]).add(multiply(x[1], y[3])),
        multiply(x[2], y[0]).add(multiply(x[3], y[2])),
        multiply(x[2], y[1]).add(multiply(x[3], y[3])) };
        }

// Multiplies two BigIntegers.
private static BigInteger multiply(BigInteger x, BigInteger y)
{
                return x.multiply(y);
        }

static BigInteger[] matrix1 =
 { BigInteger.ONE, BigInteger.ONE,
 BigInteger.ONE, BigInteger.ZERO };

static BigInteger[] matrix2 =
 { BigInteger.ONE, BigInteger.ZERO,
 BigInteger.ZERO, BigInteger.ONE };
```

1.2 Question 2 : Pour chaque méthode afficher le temps d'exécution nécessaire au calcul de chaque terme.

1.2.1 N = 0 and Eclipse IDE

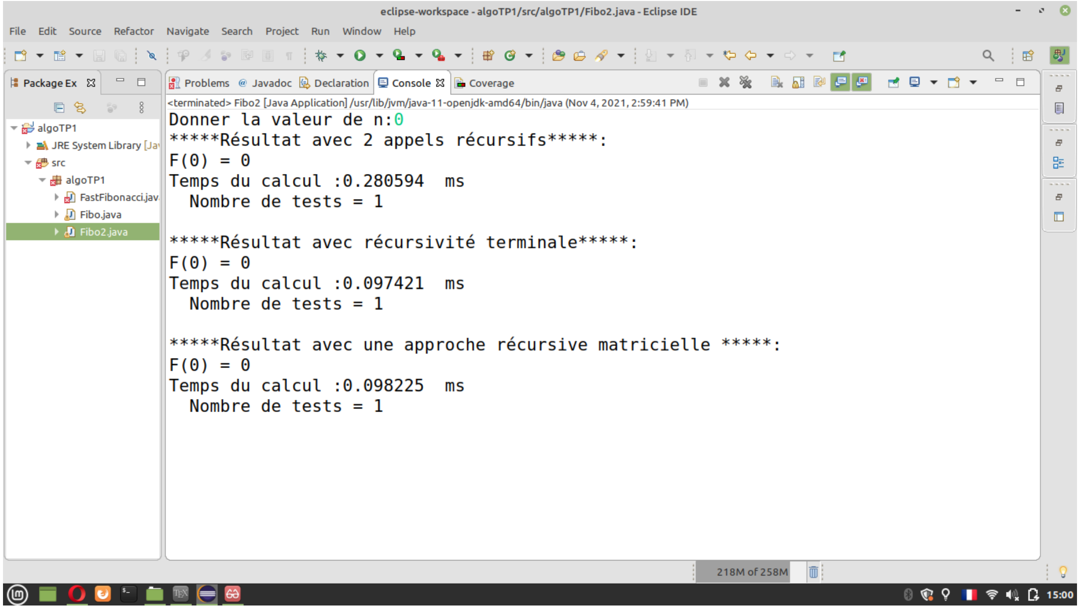

FIGURE 1.1: Value of n=0 using eclipse IDE

1.2.2 N = 1 and Eclipse IDE

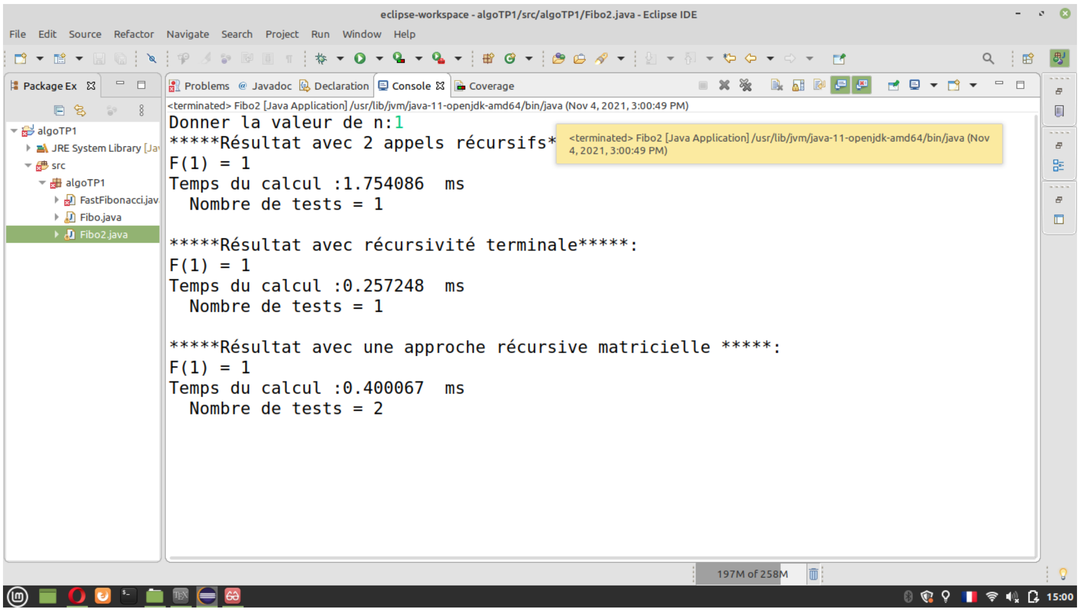

FIGURE 1.2: Value of n=1 using eclipse IDE

1.2.3 N = 10 and Eclipse IDE

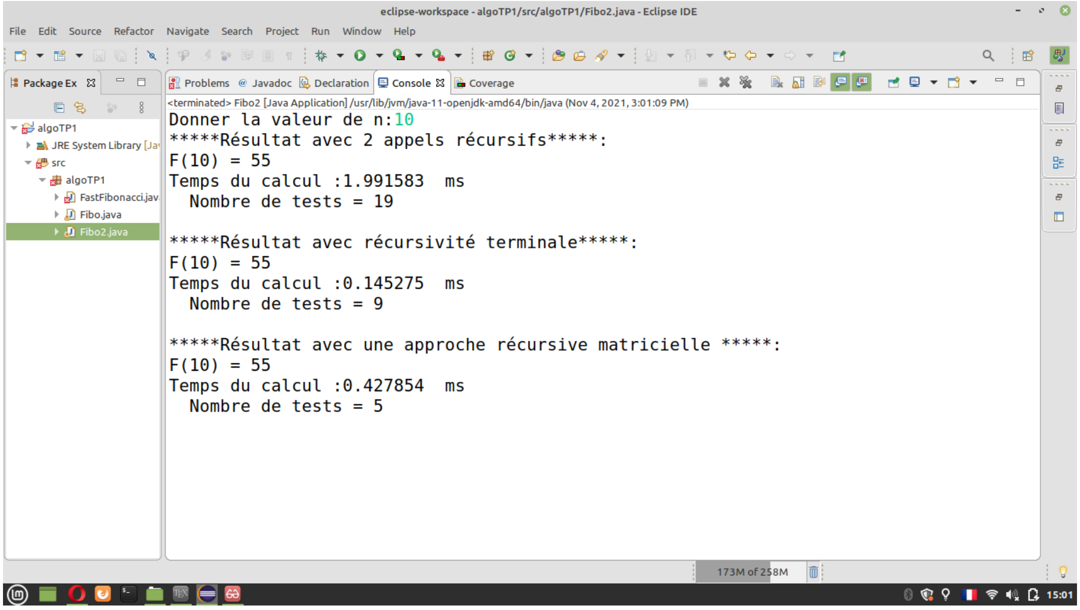

FIGURE 1.3: Value of n=10 using eclipse IDE

1.2.4 N = 80 and Eclipse IDE

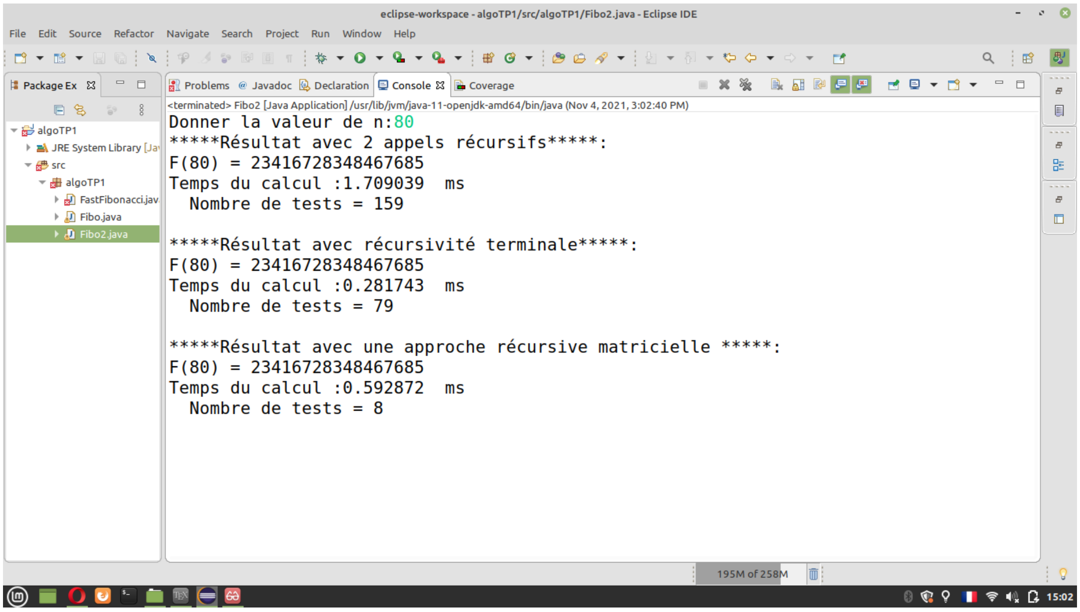

FIGURE 1.4: Value of n=80 using eclipse IDE

1.2.5 N = 100 and Eclipse IDE

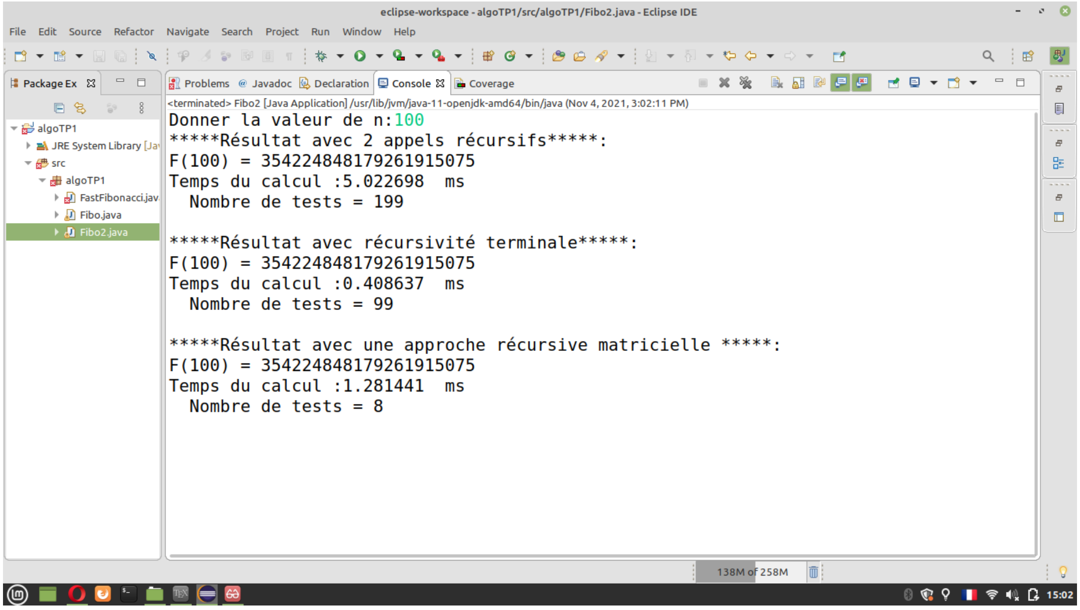

FIGURE 1.5: Value of n=100 using eclipse IDE

1.2.6 N = 10000 and Eclipse IDE

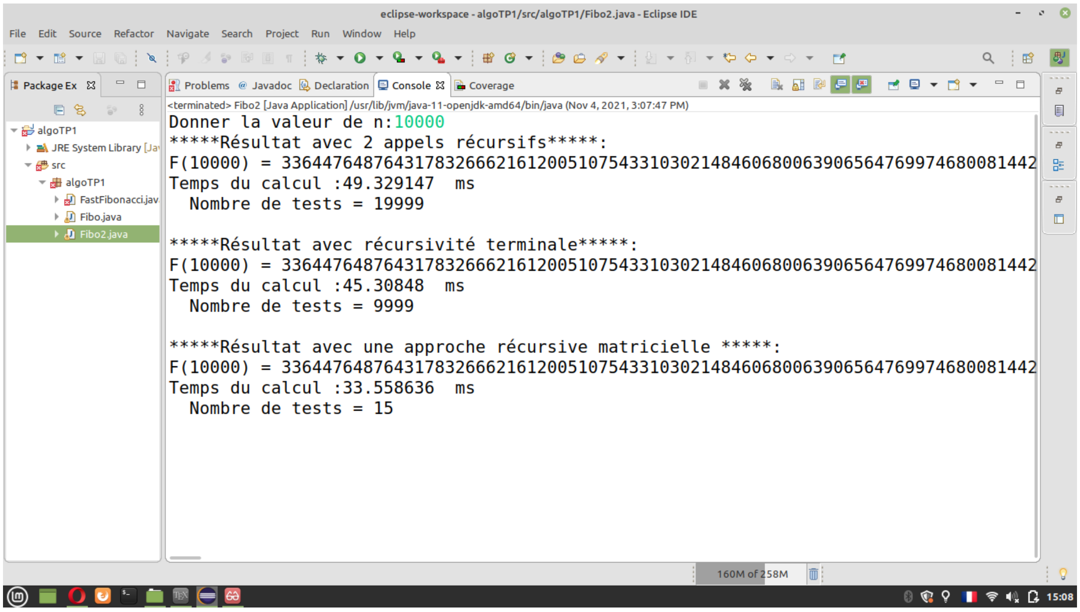

FIGURE 1.6: Value of n=10000 using eclipse IDE

1.2.7 Conclusion

As we can see from screenshots the larger we go the more efficient **Matrix method becomes**, in the last screenshot we did a test for the value of **n=10000** and the we see that it did 15 operations only compared to 9999 in second method and 19999 in the first method.

1.3 Question 3 : Afficher le nombre de tests (if (n == 0)) exécutés par chaque méthode récursive.

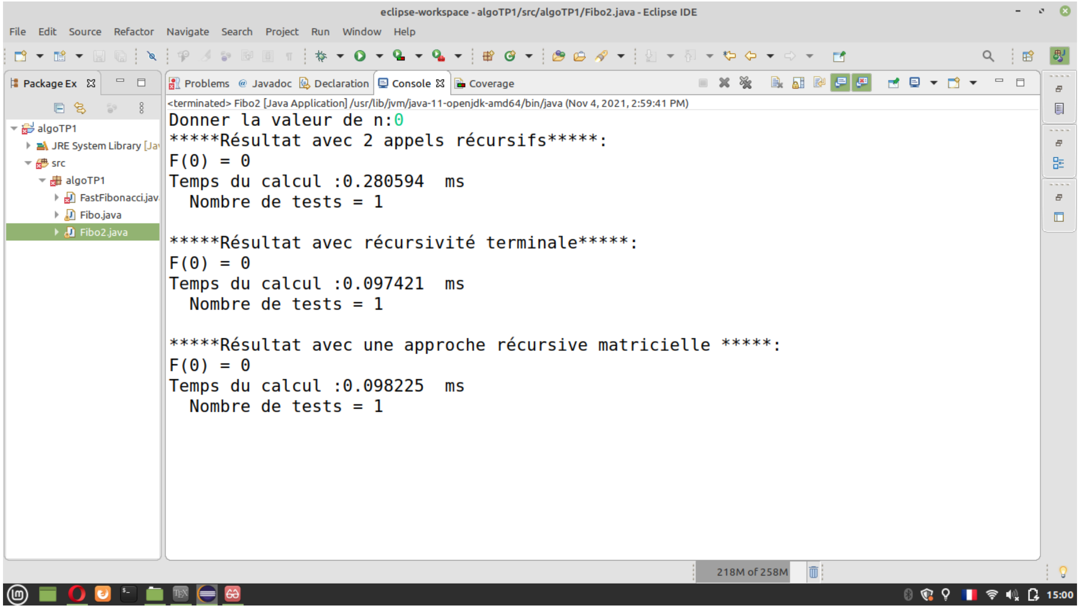

FIGURE 1.7: Value of n=0 using eclipse IDE

They are all the same with 1 test only for all 3 methods since **n = 0** is an **exit condition**.

1.4 Question 4 : Faire des captures d'écran (visibles) pour n=10 ; n=11 ; n=50 ; n=100.

1.4.1 N = 10 and Eclipse IDE

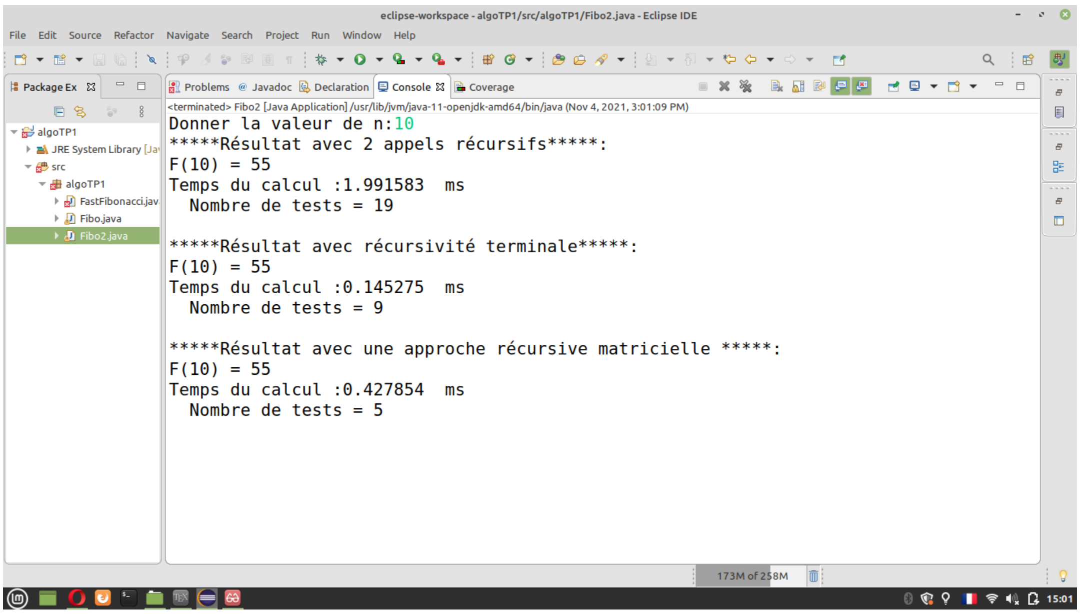

FIGURE 1.8: Value of n=10 using eclipse IDE

1.4.2 N = 11 and Eclipse IDE

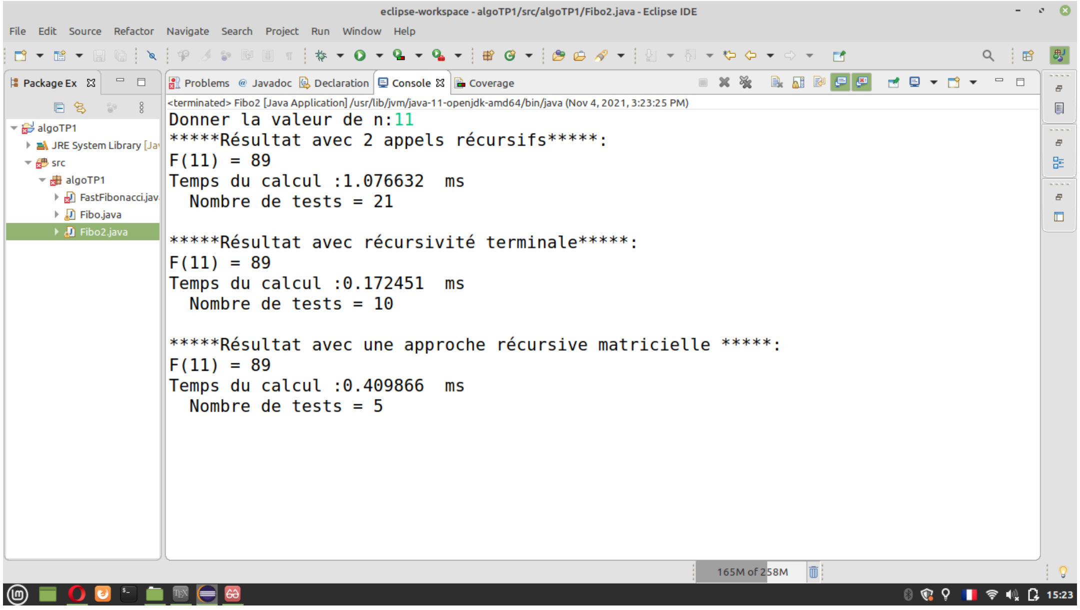

FIGURE 1.9: Value of n=11 using eclipse IDE

1.4.3 N = 50 and Eclipse IDE

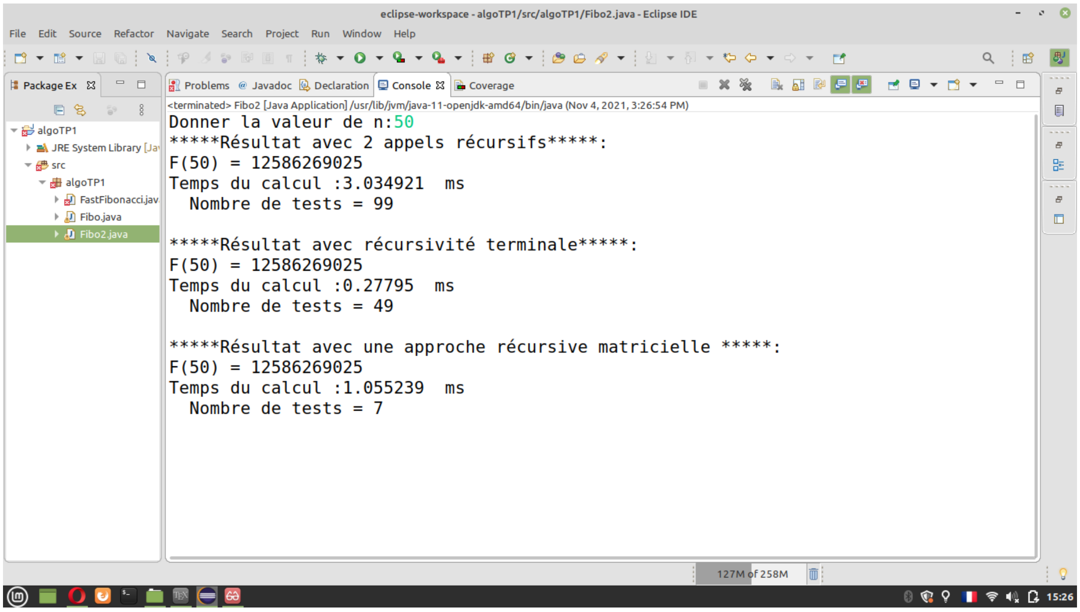

FIGURE 1.10: Value of n=50 using eclipse IDE

1.4.4 N = 100 and Eclipse IDE

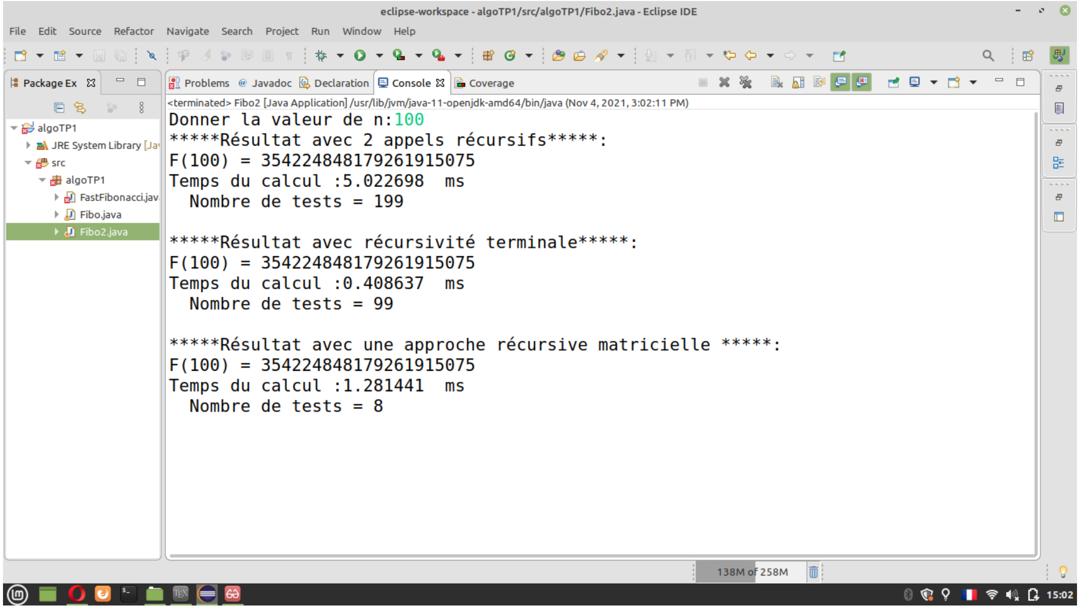

FIGURE 1.11: Value of n=100 using eclipse IDE

1.5 Question 5 : Remplir le tableau comparatif entre les 6 méthodes développées au niveau du TP1 et du TP 2.

	Methode 1	Methode 2	Methode 3	Methode 4	Methode 5	Methode 6
Valeur de F(50)	12586269025	12586269025	12586269025	12586269025	12586269025	12586269025
Temps pour calculer F(50)	4.358078ms	1.149328ms	1.49264ms	2.882976ms	0.29612ms	0.614395ms
Opération barométrique	fib[i] = fib[i - 1].add(fib[i - 2])	b = a.add(b)	fib[1] = fib[0].add(fib[1])	if (n == 0)	if (n == 0)	if (n == 0)
Nombre de fois où l'opération barométrique est exécutée pour le calcul de F(50)	49	48	48	99	49	7
Complexité temporelle	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(logn)$
Complexité spatiale	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$

1.5.1 Que peut-on remarquer ?

What we learned is that there are many ways to calculate the Fibonacci series, maybe we did only 6 but there are many more, some are good at time complexity and some are better at space complexity,

1.5.2 Que peut-on conclure ?

On general using recursion is bad without any sort of memorization technique, and iterative methods on general performs better. The Matrix method is an exception.

1.5.3 Quelle est la méthode la plus efficace ?

The matrix method was by far the best solution in terms of Time complexity, and i was surprised by the result as it has much more code and other helping functions like matrix multiplication function.

1.5.4 La moins efficace ?

Recursive functions without any memorization techniques are the worst and require exponential time as they call function for things that were already calculated.

1.5.5 Pourquoi ?

As mentioned above the process of recalculating and calling the function consumes processing time.