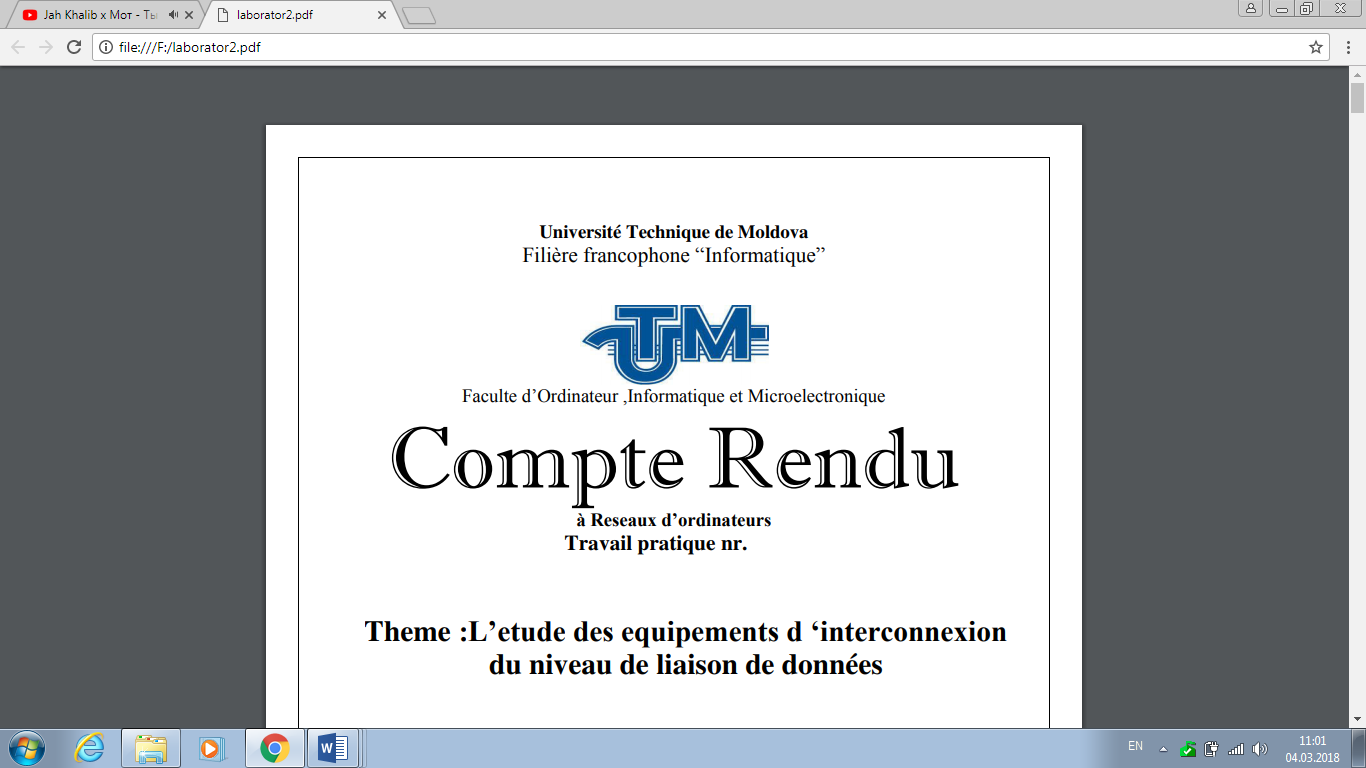
Ministère de l'éducation, de la culture et de la recherche

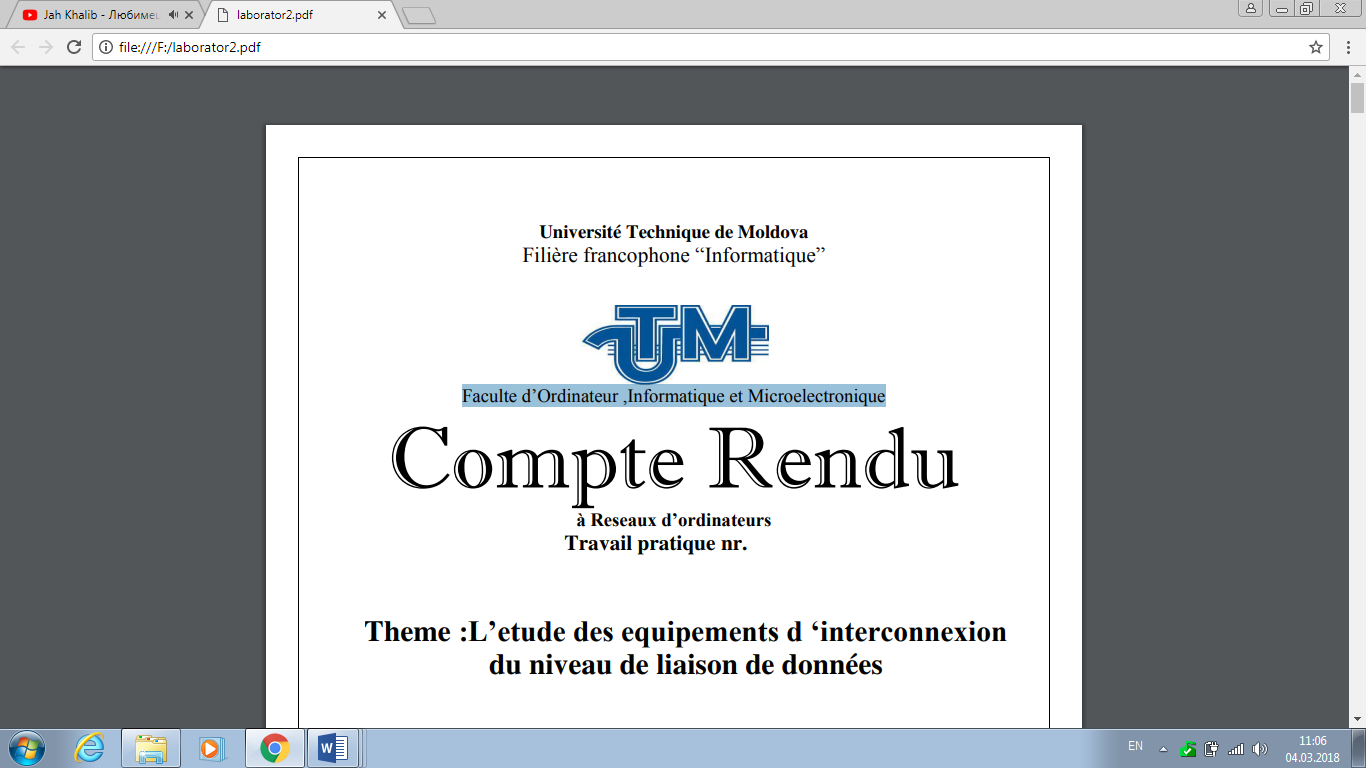
de la République de Moldova

Université technique de Moldavie

Faculté d’Ordinateur, Informatique et Microélectronique

Filière francophone “Informatique”





**TIDPP**

Travail pratique nr.4

**Thème:** **Debugging, profiling, logging.**

Effectué par l’étudiant de gr FI-181 : Damean Alexandra

Vérifié par le professeur : Rusu Viorel

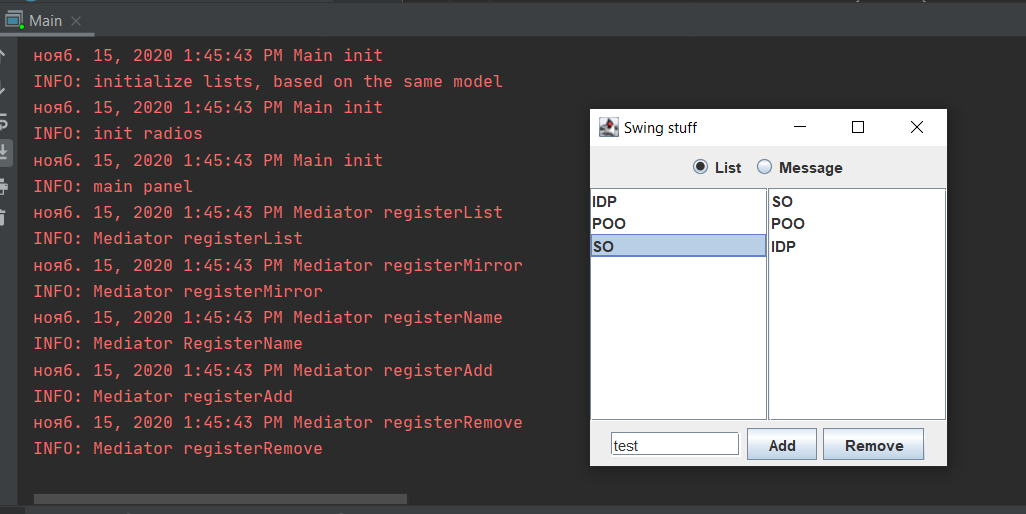
Chișinau 2020

**But:** expérimenter un mécanisme de test complémentaire, pour détecter les bogues et valider un programme ainsi que se familiariser avec les outils de débogage, étudier les performances et analyser l'exactitude du code.

1. (1p) Adaugati instructiuni de logging, cu diferite prioritati, in mai multe puncte ale programului (inclusiv in clasa ListWorker), pentru a putea urmari un lant complet de apeluri, de la interfata grafica pana la executarea actiunii (GUI - mediator - state manager etc.).

a. Botezati logger-ii cu numele claselor in care sunt definiti.

b. Afisati mai intai la consola (BasicConfigurator).



Ex :

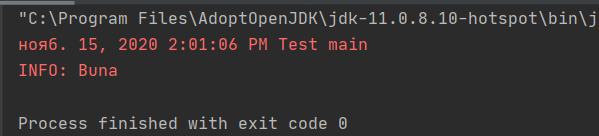
static Logger *logger* = Logger.*getLogger*("Test"); //sau Logger.getLogger("Test");

1. (1p) Creati un fisier log4j.properties, la acelasi nivel cu fiserele .class, cu continutul din textul laboratorului. Plasand fisierul in aceasta locatie, va fi citit automat dupa inlaturarea apelului BasicConfigurator.configure(). Ce reprezinta rootLogger din fisier?

Schimbati log level-ul (prioritatea) si rulati aplicatia.

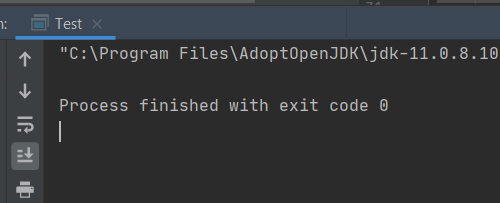


public class Test {  
 static Logger *logger* = Logger.*getLogger*("Test"); //sau Logger.getLogger("Test");  
  
  
 public static void main(String[] args) {  
 PropertyConfigurator.*configure*("log4j.properties");  
 *logger*.setLevel(Level.*ALL*);  
 *logger*.info("Buna");  
 }



Modificat:

public class Test {  
 static Logger *logger* = Logger.*getLogger*("Test"); //sau Logger.getLogger("Test");  
  
  
 public static void main(String[] args) {  
 **PropertyConfigurator**.*configure*("log4j.properties");  
 *logger*.setLevel(Level.*WARNING*);  
 *logger*.info("Buna");  
 }

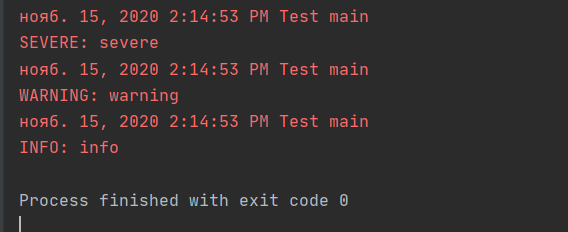


1. (1p) Modificati fisierul de configurare, prin adaugarea un file appender (manual log4j). Rulati.
2. Adaugati un pattern layout pentru noul appender, asemanator cu layout-ul pentru ConsoleAppender, din fisier, si tipariti numele fisierului, numele metodei si linia din fisier (cititi despre PatternLayout). Studiati fisierul de log.

log4j.rootLogger=debug, stdout, R  
  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
  
# Pattern to output the caller's file name and line number.  
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n  
  
log4j.appender.R=org.apache.log4j.RollingFileAppender  
log4j.appender.R.File=example.log  
  
log4j.appender.R.MaxFileSize=100KB  
# Keep one backup file  
log4j.appender.R.MaxBackupIndex=1  
  
log4j.appender.R.layout=org.apache.log4j.PatternLayout  
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n

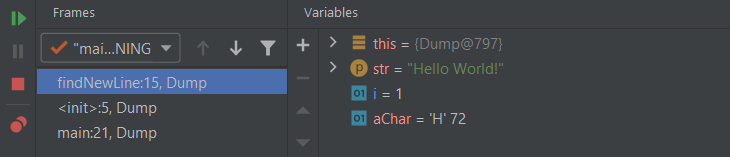
1. (1p) Adaugati un rolling file appender, ce permite generarea unor fisiere de log de o dimensiune maxima, urmand ca, la depasirea acesteia, sa se comute la alt fisier de log. De asemenea, permite precizarea numarului maxim de astfel de fisiere, ce se doresc pastrate. a. Stabiliti, doar pentru acest appender, un prag de logare mai mare. Hint: parametrul Threshold

import java.io.IOException;  
import java.util.logging.FileHandler;  
import java.util.logging.Level;  
  
import org.apache.log4j.BasicConfigurator;  
import org.apache.log4j.PropertyConfigurator;  
import java.util.logging.Logger;  
import java.util.logging.SimpleFormatter;  
  
public class Test {  
 static Logger *logger* = Logger.*getLogger*("Test"); //sau Logger.getLogger("Test");  
  
 public static void main(String[] args) {  
  
 Logger logger = Logger.*getLogger*("Test");  
 FileHandler fh;  
  
 try {  
  
 // This block configure the logger with handler and formatter  
 fh = new FileHandler("D:/Documents/TIDPP/Lab4/TIDPP\_Lab\_4a\_skel/LoggindOutput/log.txt", 3000, 3);  
 logger.addHandler(fh);  
 logger.setLevel(Level.*SEVERE*);  
 SimpleFormatter formatter = new SimpleFormatter();  
 fh.setFormatter(formatter);  
  
 // the following statement is used to log any messages  
 logger.info("My first log");  
  
 } catch (SecurityException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 logger.severe("severe");  
 logger.warning("warning");  
 logger.info("info");  
 logger.config("config");  
 logger.fine("fine");  
 logger.finer("finer");  
 logger.finest("finest");  
// }  
 }  
}

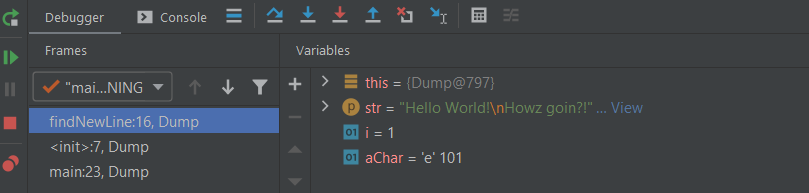


**Debugging & Thread dumping**

public class Dump {  
   
 public Dump(String str) {  
 System.*out*.println("New line at " + findNewLine(str));  
 }  
   
 private int findNewLine(String str) {  
 int i = 0;  
 char aChar;  
   
 do {  
 aChar = str.charAt(i);  
 i++;  
 } while(aChar != '\n');  
   
 return i;  
 }  
  
 public static void main(String[] args) throws Exception {  
 new Dump("Hello World!");  
 }

****

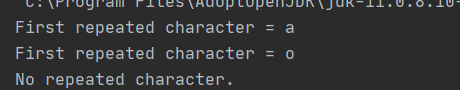
1. Pe masura ce parcurgeti instructiunile, incercati sa: o editati valorile variabilelor direct in tabelul de variabile (inclusiv membrul value al lui str) o porniti urmarirea expresiei str.charAt(i) (selectare si alegere Watch din meniul contextual) o rezolvati problema astfel incat programul sa ajunga la final fara blocare

****

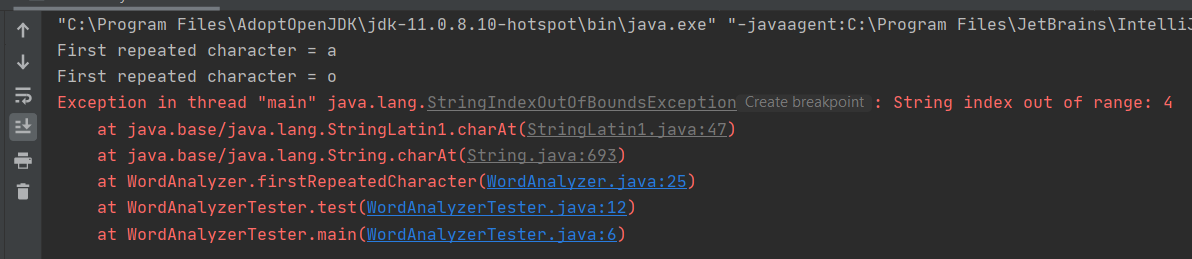
public class Dump {  
   
 public Dump(String str) {  
 System.*out*.println("New line at " + findNewLine(str));  
 }  
   
 private int findNewLine(String str) {  
 int i = 0;  
 char aChar;  
   
 do {  
 aChar = str.charAt(i);  
 i++; // I nu se incrementa. Se verifica mereu str.chatAt(0)  
 } while(aChar != '\n');  
   
 return i;  
 }  
  
 public static void main(String[] args) throws Exception {  
 new Dump("Hello World!\nHowz goin?!");  
 }  
  
}

1. Analizati clasele WordAnalyzer si WordAnalyzerTester. Exemplul construieste un analizor de cuvinte, clasa WordAnalyzer, si un test pentru aceasta clasa.

public char firstRepeatedCharacter() {  
 for (int i = 0; i < word.length() - 1; i++)  
 {  
 char ch = word.charAt(i);  
 if (ch == word.charAt(i + 1))  
 return ch;  
 }  
 return 0;  
}  
  
*/\*\**

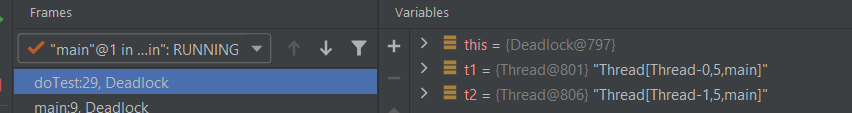
****

public char firstRepeatedCharacter() {  
 for (int i = 0; i < word.length(); i++)  
 {  
 char ch = word.charAt(i);  
 if (ch == word.charAt(i + 1))  
 return ch;  
 }  
 return 0;  
}

****

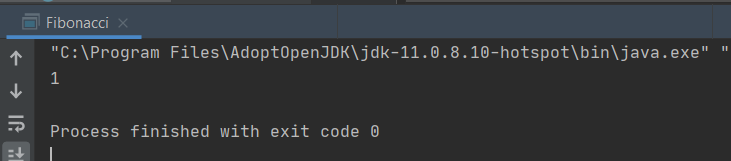
**2(0.5p). Deadlock debugging**

public class Deadlock {  
 private static int *sleepMillis*=5;  
 private final Object lock1 = new Object();  
 private final Object lock2 = new Object();  
   
 public static void main(String[] args) {  
   
 Deadlock test = new Deadlock();  
 test.doTest();  
 }  
   
 private void doTest() {  
 Thread t1 = new Thread(new Runnable() {  
 public void run() {  
 System.*out*.println("Thread 1");  
// lock12();  
   
 }  
 });  
 Thread t2 = new Thread(new Runnable() {  
 public void run() {  
 System.*out*.println("Thread 2");  
// lock21();  
   
 }  
 });  
 t1.start();  
 t2.start();  
 System.*out*.println("Main thread");  
 }  
   
 private void lock12() {  
 synchronized (lock1) {  
 sleep();  
 synchronized (lock2) {  
 sleep();  
 }  
 }  
 }  
   
 private void lock21() {  
 synchronized (lock2) {  
 sleep();  
 synchronized (lock1) {  
 sleep();  
 }  
 }  
 }  
   
 private void sleep() {  
 try {  
 Thread.*sleep*(*sleepMillis*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

****

**3(2p). Debugging**

import java.io.\*;  
  
public class Fibonacci {  
 */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The Fibonacci function will return the num'th Fibonacci number \*  
 \* F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2) \*  
 \* PRE: num >= 0 \*  
 \* POST: result = F(num) \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* public static int fibonacci (int num) {  
 int calc = 0; // The value of num! to be returned  
  
 if (num <= 1) // F(0) = 0, F(1) = 1, assume num >= 0  
 calc = num;  
 else // num > 1  
 calc = *fibonacci* (num - 1) + *fibonacci* (num - 2); // F(n) = F(n-1) + F(n-2)  
  
 return calc;  
 }  
  
 */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* The main routine \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* public static void main (String [] args) {  
 int number = 0; // The number read in  
  
 // See if we have the correct number of input parameters  
 if (args.length != 1) {  
 System.*out*.println ("You must provide a paramater of 1 number.");  
 System.*exit* (-1);  
 }  
  
 // Try to convert the parameter to a number  
 try {  
 number = Integer.*parseInt* (args [0]);  
 } catch (NumberFormatException x) {  
 System.*out*.println ("The parameter must be a number.");  
 System.*exit* (-1);  
 }  
  
 // Check that it is at least 0  
 if (number < 0) {  
 System.*out*.println  
 ("Can only calculate non-negative Fibonacci number.");  
 System.*exit* (-1);  
 }  
  
 // Calculate the result  
   
 System.*out*.println (*fibonacci* (number));  
 }  
  
  
}

****

**Conclusion :**

Le but de ce travail pratique était passer les étapes de test et de déploiement dans le cycle de développement d'un programme et en utilisant les outils Java spécifiques à ces étapes. Je me suis familiarisé avec l'outil JUnit. On a etudie la methode Black-box testing. Je me suis familiarisé avec l'outil Dependency Injection.

Au bout de tout on peut dire que l’etape de controle est tres importante, parce que cela peut éviter un risque coûteux à l'avenir.