

Oblig 5 Hashing

2016

Høgskolen i Narvik

ITE1806 – Datakommunikasjon & Sikkerhet

Allan Arnesen 140518

Innholdsfortegnelse

1	Klargjøring.....	3
2	Planlegging av programmet.....	3
3	Selve programmet.....	4
3.1	WhereAllTheMagicHappens.java	4
3.2	GVH.java.....	10
3.3	TheGUI.java.....	11
4	Hvordan bruke programmet.....	13
4.1	I GUI modus	13
4.2	Via kommandolinje	15
5	Testkjøring.....	16
6	Problemer og utfordringer	18
	Referanseliste	19
	Vedlegg	19

Tabelliste

Tabell 1 - Resultat av testkjøring av programmet.	17
--	----

Figurliste

Figur 1 - Starten av WhereAllTheMagicHappens.java	4
Figur 2 - Metoder for å sette mode og få mode. Disse to metodene er vel selvforklart. Benyttes til å sette mode (CMD eller GUI) og returnere hvilken mode som benyttes nå.	4
Figur 3 - getFilesAndGenerateHash() del 1	5
Figur 4 - getFilesAndGenerateHash() del 2	6
Figur 5 - getFilesAndVerifyHash() kalles i verifyThatHashIsValid()	6
Figur 6 - makehash()	7
Figur 7 - writeHashToFile()	8
Figur 8 - checkIfFileIsOnTheList()	8
Figur 9 - getVer() og getHash().....	9
Figur 10 - setHashFile() og clearGUIResponse().....	9
Figur 11 – getAnswerForGUIVer().....	9
Figur 12 – getAnswerForGUIhash()	9
Figur 13 - GVH.java.....	10
Figur 14 - Starten av TheGUI.java.....	11
Figur 15 - btHash & btVer	12
Figur 16 – TheGUI	13
Figur 17 - Verifiserer filer.....	13
Figur 18 - Eksempel hashing, men filene er allerede hashet.....	14

Figur 19 - Eksempel hashing.....	14
Figur 20 - Starte GUI fra kommandolinje	15
Figur 21 - Eksempel på verifisering i kommandolinje	16
Figur 22 - Eksempel på hashing i kommandolinje	16

Forord

I denne oppgaven skal jeg lage et program som kan sjekke integriteten til en fil eller innholdet i en mappe. Programmet skal kunne ta imot argument som fil eller dir som skal hashes, for så å generere hash til filene, og lagre dette i en egen fil. Programmet skal også kunne ta imot en fil eller en dir og en fil som holder hash verdier og verifisere at filen(e) ikke er endret siden de ble opprettet(hashet). Hash som skal benyttes er sha256.

Jeg velger å lage et program som man kan benytte via GUI samt via kommando linjen.

Denne rapporten vil inneholde forklaring for koden, hvordan programmet brukes og valgene jeg har tatt. Det er også forklaring i selve koden for stort sett samtlige kodelinjer som er lagt til. Ved spørsmål til spesifikke kodelinjer, se selve javafilen. Der er det utdypende kommentarer til hver kodelinje.

1 Klargjøring

Det første jeg gjør er å importere kodene som er oppgitt i oppgaven. Jeg henter programmet fra Tanenbaum¹. Dette programmet kan traversere gjennom en mappe og undermapper og finne tak i alle filene som ligger der. Det neste jeg gjør er å legge til commons-codec-1.10² som er oppgitt i oppgaveteksten³. Dette er biblioteket som tar seg av selve hashingen. Her fulgte jeg fremgangsmåten som oppgitt i teksten.

Når dette er gjort starter jeg med kodingen.

2 Planlegging av programmet

Til å begynne med lager jeg to filer. En fil som skal ta seg av å lage hash til filene, og skrive dette til en fil. Disse filene ble kalt MakeHash.java og VerHash.java. Planen var opprinnelig å lage kun et kommandobasert program. Begge disse klassene skulle styres av en fil som het MyHashProg.java

Normalt sett når jeg skal lage et program har jeg lagt en plan om hvordan programmet skal være laget, med metoder og klasser klart for meg. Men i denne oppgaven valgte jeg å «kode fritt» siden det var snakk i et ganske lite og enkelt program. Det ble dermed for endret bort fra å ha et program for å lage hash og et program for å verifisere hash til at jeg skulle lage en GUI som styrte alt fra samme grensesnitt. Og helst med så lite knapper og valg som mulig. Men du skal også kunne kjøre programmet fra kommandolinjen.

Jeg samlet så alle metodene mine i en fil kalt WhereAllTheMagicHappens.java. Dette er filen som både genererer og verifiserer hash verdier mot filene. Det er også laget en fil som heter TheGUI.java som da holder alt for selve GUI'en og en fil som heter GVH.java som tar imot argumenter fra kommandolinjen.

For selve kodingen har vi allerede dekt og vært gjennom all kode tidligere som trengs for å fullføre denne oppgaven, så innhenting av nye kunnskaper regner jeg med vil være et minimum. Men i de tilfellene jeg har måtte oppsøke hjelp vil dette være beskrevet.

¹ <http://kark.hin.no/opsys/tanenbaum/java/ExecutableFiles.java>

² http://kark.hin.no/sikkerhet/filehash/oblig5_tips.html

³ <https://hin.itslearning.com/ContentArea/ContentArea.aspx?LocationID=4387&LocationType=1&ElementID=429779>

3 Selve programmet

Gjennomgang av javafilene mine – et underkapittel for hver fil. Du gjøres også oppmerksom på at alle javafiler har en mer detaljert beskrivelse som kommentarer i selve javafilen.

3.1 WhereAllTheMagicHappens.java

Vi går rett til flesket og går gjennom selve koden. Vi starter med hovedfilen –

WhereAllTheMagicHappens.java. Jeg vil nå gå gjennom dette programmet skritt for skritt.

Vi starter på toppen.

```
17 public class WhereAllTheMagicHappens {
18
19     private static String[] temp_hit_on_file;
20     public static ArrayList<String> returnStringForGui = new ArrayList<>();
21     private final static boolean GUI = true;
22     private final static boolean CMD = false;
23     private static String HashFile;
24
25     // Set mode here
26     public static boolean mode = GUI;
27
28
29
```

Figur 1 - Starten av WhereAllTheMagicHappens.java

Dette er starten av WhereAllTheMagicHappens.java filen. Her definerer vi selve klassen og lager oss noen holdere for å holde på diverse informasjon.

Temp hit on file - holder på eventuelle filer vi finner som allerede finnes i hash filen. Denne benyttes for å lete etter duplikater og for å returnere eventuelle filnavn som er duplikat.

returnStringForGui – er en array som holder alle svar som kommer fra programmet. Denne benyttes av GUI delen for å kunne printe ut svarene fra programmet.

GUI/CMD – Holder henholdsvis true eller false. Benyttes for å fortelle programmet hvilken måte svar skal gis. Kommandolinje eller GUI.

mode – holder på GUI eller CMD. Forteller programmet hvilken modus som nå benyttes.

```
29
30     public static void SetMode (boolean wichMode) {
31         mode = wichMode;
32     }
33     public static boolean getMode () {
34         return mode;
35     }
```

Figur 2 - Metoder for å sette mode og få mode. Disse to metodene er vel selvforklart. Benyttes til å sette mode (CMD eller GUI) og returnere hvilken mode som benyttes nå.

```

36 public static void getFilesAndGenerateHash(File directory){
37     File entry;
38     String entryName;
39
40     ArrayList<File> files = new ArrayList<>();
41
42     if(directory.isDirectory()){
43         if(mode){ // If GUI are the mode Write this to the return array to the GUI
44             returnStringForGui.add("Starting search of directory "
45                                     + directory.getAbsolutePath());
46         }
47         else { // Else use sysout
48             System.out.println("Starting search of directory "
49                                 + directory.getAbsolutePath());
50         }
51         String contents[] = directory.list();
52         if(contents == null) return;
53
54         for(int i=0; i<contents.length; i++){
55             entry = new File(directory, contents[i]);
56
57             if(contents[i].charAt(0) == '.')
58                 continue;
59
60             if (entry.isDirectory()){
61                 getFilesAndGenerateHash(entry);
62             }
63             else {
64                 if(entry.isFile())
65                     files.add(entry);
66             }
67         }
68     }
69     writeHashToFile(makehash(files));
70 }

```

Figur 3 - getFilesAndGenerateHash() del 1

Metoden getFilesAndGenerateHash er metoden som jobber seg gjennom mappen/filen som er presentert. Den jobber seg rekursivt gjennom alle mapper og legger alle filene den finner til en array kalt files, med unntak av . og .. dir. Når alle filene er funnet sender den alle filene til metoden writeHashToFile. Her ser du også at utskriften bestemmes av hvilken mode vi er i. Enten skrives svaret direkte til konsoll(CMD) eller så lagres svaret til returnStringForGui arrayet som returneres til slutt til TheGUI.java.

```

71     else if(directory.isFile()){
72         if(mode){ // If GUI are the mode Write this to the return array to the GUI
73             returnStringForGui.add("Starting the check on file "
74                 + directory.getAbsolutePath());
75         }
76         else { // else use sysout
77             System.out.println("Starting the check on file "
78                 + directory.getAbsolutePath());
79         }
80         files.add(directory);
81         writeHashToFile(makehash(files));
82     }
83     else {
84         if(mode){ // If GUI are the mode write this to the return array to the GUI
85             returnStringForGui.add("Could not determin if this is a file/directory or if the
86             returnStringForGui.add("Remember to add the file extention if it is a file.");
87         }
88         else { // else use sysout
89             System.out.println("Could not determin if this is a file/directory or if the fil
90             System.out.println("Remember to add the file extention if it is a file.");
91         }
92     }
93 }
94 if(mode){ // If GUI are the mode write this to the return array to the GUI
95     returnStringForGui.add("Done with " + directory.getName() + "\n");
96 }
97
98 }

```

Figur 4 - getFilesAndGenerateHash() del 2

Om metoden finner ut at «filen» vi har gitt den ikke er en dir, men en enkelt fil orker vi ikke å lete så mye og sender filen direkte inn til writeHashToFile. Men MERK at filene først kjøres gjennom en metode kalt makehash.

PS. Metoden getFilesAndVerifyHash gjør akkurat det samme som denne metoden, bare det at den sender filene/filen til en metode kalt verifyThatHashIsValid. Jeg går ikke gjennom den metoden her, men vil gjennomgå verifyThatHashIsValid litt lengre ned. Se fig 5.

```

123
124         if (entry.isDirectory()) {
125             getFilesAndVerifyHash(entry);
126         }

```

Figur 5 - getFilesAndVerifyHash() kalles i verifyThatHashIsValid()

Bilde bare for å vise forskjellen. Disse metodene kunne egentlig godt vært slått sammen og dermed spart litt plass i koden.

```

223 public static ArrayList<String> makehash(ArrayList<File> files){
224     String apache_sha256="";
225     File fileToHash;
226     ArrayList<String> returnValues = new ArrayList<>();
227     for(int i = 0; i < files.size(); i++){
228         fileToHash = files.get(i);
229         try {
230             FileInputStream in = new FileInputStream(fileToHash);
231             apache_sha256 = DigestUtils.sha256Hex(in);
232         }
233         catch (FileNotFoundException ex) {
234             if(mode){ // If GUI are the mode write this to the return array to the GUI
235                 returnStringForGui.add("File not found");
236             }
237             else { // Else use sysout
238                 System.out.println("File not found");
239             }
240         }
241         catch (IOException ex) {
242             if(mode){ // If GUI are the mode write this to the return array to the GUI
243                 returnStringForGui.add("Something went wrong...");
244             }
245             else{ // Else use sysout
246                 System.out.println("Something went wrong...");
247             }
248         }
249     }
250
251     if(checkIfFileIsOnTheList(fileToHash) == false){
252         returnValues.add(fileToHash + " " + apache_sha256);
253     }
254     else {
255         if(mode){ // If GUI are the mode write this to the return array to the GUI
256             returnStringForGui.add("The file " + fileToHash.getName() + " is already has");
257         }
258         else{ // Else use sysout
259             System.out.println("The file " + fileToHash + " is already hashed");
260         }
261     }
262 }
263
264 }
265
266 return returnValues;
267 }

```

Figur 6 - makehash()

Denne metoden tar imot alle filene vi har funnet. Den leser så av hver enkelt fil og genererer en hash til filen. Vi tar så en enkel sjekk om filen allerede er lagret i hashfilen vår (altså om vi allerede har en hash til denne filen). Har vi ikke denne filen fra før lager vi en string som består av filnavnet, et mellomrom og så selve hashen. Dette legger vi til arrayet returnValues. Det er dette arrayet som igjen sendes videre til writeHashToFile. Vi skriver i tillegg en setning til enten returnStringForGui eller cmd med status for makehash – men kun om filen allerede eksisterer i hashlisten vår.


```

293 public static void writeHashToFile(ArrayList<String> filesAndHash) {
294     try(FileWriter fw = new FileWriter(HashFile, true); // Start
295         BufferedWriter bw = new BufferedWriter(fw);
296
297         PrintWriter out = new PrintWriter(bw)){
298             for(int i = 0; i < filesAndHash.size(); i++){
299                 out.println(filesAndHash.get(i));
300                 if(mode){ // If GUI are the mode write this to the return array to the GUI
301                     returnStringForGui.add(filesAndHash.get(i));
302                 }
303                 else{ // Else use svsout
304                     System.out.println(filesAndHash.get(i) + " is hashed");
305                 }
306             }
307         }
308     catch (IOException e) {
309         if(mode){ // If GUI are the mode write this to the return array to the GUI
310             returnStringForGui.add("Something went wrong....");
311         }
312         else{ // Else use svsout
313             System.out.println("Something went wrong...");
314         }
315     }
316 }
317
318 }

```

Figur 7 - writeHashToFile()

I denne metoden tar vi så imot arrayet returnValues og jobber oss gjennom den linje for linje. For hver linje vi har, så legger vi den til i ønsket hashfil. Vi skriver i tillegg en linje enten til returnStringForGui eller til kommandolinjen med status.

```

270 private static boolean checkIfFileIsOnTheList(File fileToCheck){
271     boolean isPresent = false;
272     File hashfile = new File(HashFile);
273     String toCheck = fileToCheck.getAbsolutePath();
274     String[] line;
275     Scanner scan;
276     try {
277         scan = new Scanner(hashfile);
278         while(scan.hasNextLine()){
279             line = scan.nextLine().split(" ");
280             if(toCheck.contains(line[0])){
281                 temp_hit_on_file = line;
282                 isPresent = true;
283             }
284         }
285         scan.close();
286     } catch (FileNotFoundException e) {
287
288         //System.out.println("File holding hash values not found - Making theHash
289     }
290     return isPresent;
291 }

```

Figur 8 - checkIfFileIsOnTheList()

Her sjekker vi om vi finner en fil med samme navn i hashlisten vår. Om vi får en match, setter vi denne linjen til temp_hit_on_file. Her sjekker vi en og en fil som sendes inn fra makehash metoden. Vi setter også isPresent til true – retur verdien vår. Ellers er isPresent false.

```

344 public static void getVer(File fileToWorkWith, String hash){
345     SetMode(CMD);
346     setHashFile(hash);
347     getFilesAndVerifyHash(fileToWorkWith);
348 }
349 public static void getHash(File fileToWorkWith, String hash){
350     SetMode(CMD);
351     setHashFile(hash);
352     getFilesAndGenerateHash(fileToWorkWith);
353 }

```

Figur 9 - getVer() og getHash()

Disse to metodene er det som starter alt for kommandolinje versjonen. getHash eller getVer, alt etter hvilken argumenter du passerer til programmet i kommandolinjen, setter først modusen til at responsen skal være til kommandolinjen, så sender den filene du har sagt skal holde/holder hashverdiene til en metode setHashFile(se fig 10). Så til slutt sender den filen/dir vi skal jobbe med til metoden for å rekursivt finne alle filene og gjøre alt det som er beskrevet til nå.

```

325 private static void setHashFile(String hash) {
326     if(hash.isEmpty()){
327         returnStringForGui.add("No hash file is supplied.\n"
328             + "Making the file TheHashList.txt\n"
329             + "ONLY IF YOU ARE MAKING HASHES - ELSE WE TRY TO FIND THAT FILE");
330         HashFile = "TheHashList.txt";
331     }
332     else HashFile = hash;
333 }
334 }
335 public static void clearGUIResponse() {
336     returnStringForGui.clear();
337 }

```

Figur 10 - setHashFile() og clearGUIResponse()

setHashFile er en metode som kun sjekker at vi har lagt inn et ønsket filnavn og oppdater HashFile med denne verdien. Har du ikke lagt inn noen hashfil og du skal generere hash, vil hash bli skrevet til en default fil med navn TheHashList.txt.

Metoden clearGUIResponse gjør akkurat det. Den tømmer arrayet som holder på returverdiene til GUIen. Dette i tilfelle du vil kjøre programmet flere ganger. Vi trenger da ikke de gamle verdiene som vi er ferdige med.

```

319 public static ArrayList<String> getAnswerForGUIVer(File fileToWorkWith, String hash) {
320     SetMode(GUI);
321     setHashFile(hash);
322     getFilesAndVerifyHash(fileToWorkWith);
323     return returnStringForGui;
324 }

```

Figur 11 – getAnswerForGUIVer()

```

338 public static ArrayList<String> getAnswerForGUIhash(File fileToWorkWith, String hash) {
339     SetMode(GUI);
340     setHashFile(hash);
341     getFilesAndGenerateHash(fileToWorkWith);
342     return returnStringForGui;
343 }

```

Figur 12 – getAnswerForGUIhash()

Disse to metodene gjør i praksis det samme som getVer og getHash. Men disse returnerer i tillegg arrayet returnStringForGui med alle svar programmet har generert til brukeren.

3.2 GVH.java

Denne filen tar i realiteten kun imot argumenter fra kommandolinjen og sender de inn til WhereAllTheMagicHappens.java.

```
4 public class GVH{
5     public static void main(String args[]) {
6
7         if(args.length == 3){
8             if(args[0].compareTo("ver") == 0){
9                 File fileToWorkWith = new File(args[1]);
10                WhereAllTheMagicHappens.getVer(fileToWorkWith, args[2]);
11            }
12            else if(args[0].compareTo("hash") == 0){
13                File fileToWorkWith = new File(args[1]);
14                WhereAllTheMagicHappens.getHash(fileToWorkWith, args[2]);
15            }
16        }
17        else{ // If we recived 3 arguments, but they where wrong
18            System.out.println("To hash/ver hash use: ver/hash file/dirToW
19        }
20    }
21    else{ // If we recived less than/or more than 3 arguments
22        System.out.println("To hash/ver hash use: ver/hash file/dirToW
23    }
24
25    // Job done!
26    System.out.println("Done");
27
28 }
```

Figur 13 - GVH.java

Vi starter med å sjekke at vi faktisk har mottatt 3 argumenter. Har vi ikke det sendes du rett til en utskrift som forklarer hvordan du må legge inn argumentene dine. Merk her at vi krever en hashfil uansett om du skal generere eller verifisere en hash. Dette for å gjøre det enkelt – og litt grunnet «latskap». Hoveddelen av utviklingen har gått til GUI kontroll over programmet.

Det neste vi gjør er å sjekke det første argumentet mot enten «ver» eller «hash» som er de to oppgavene du kan starte her. Så sendes de to neste argumentene inn til WhereAllTheMagicHappens og tilhørende metoder basert på ditt valg. De to andre argumentene er da fil/dir som skal jobbes med og fil som holder/skal holde hash.

Så tilslutt skriv ut «Done» som bekrefter at programmet er ferdig.

3.3 TheGUI.java

Dette dokumentet tar seg av selve GUI biten. Det er ingen kode her heller som har noe med selve hashingen å gjøre, på lik linje som GVH.java. Den styrer kun det grafiske. Men vi går gjennom den også. Vi starter øverst.

```
21 public class TheGUI extends Application {
22     @Override // Override the start method in the Application class
23     public void start(Stage primaryStage) {
24
25         GridPane MainPane = new GridPane();
26         MainPane.setAlignment(Pos.CENTER);
27         MainPane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
28         MainPane.setHgap(5.5);
29         MainPane.setVgap(5.5);
30         MainPane.add(new Label("Enter file or directory path:"), 0, 0);
31         TextField directory = new TextField();
32         directory.setPrefWidth(450);
33         MainPane.add(directory, 1, 0);
34         MainPane.add(new Label("Enter path to hashfile:"), 0, 1);
35         TextField hashFile = new TextField();
36         hashFile.setPrefWidth(450);
37         MainPane.add(hashFile, 1, 1);
38
39
40         FlowPane btpane = new FlowPane();
41         Button btHash = new Button("Hash files");
42         MainPane.add(btHash, 0, 2);
43         GridPane.setHalignment(btHash, HPos.RIGHT);
44         Button btVerify = new Button("Verify files");
45         MainPane.add(btVerify, 0, 2);
46         GridPane.setHalignment(btVerify, HPos.LEFT);
47         MainPane.getChildren().addAll(btpane);
48     }
```

Figur 14 - Starten av TheGUI.java

All kode du ser over her gjør i realiteten svært lite. Vi starter med å lage oss en MainPane vi kan putte alt vi skal vise frem inn i. Så gjøre vi noen stil settinger – Derav alle set.. kommandoene. Vi setter alignment, padding, horisontal og vertikal mellomrom.

Så lager vi oss en label med tilhørende tekstfelt for å ta imot filnavn/dir navn vi skal jobbe med. Vi gjør det samme for et felt for å ta imot navnet til filen som skal/holder hash.

Det neste vi gjør er å lage to knapper. En knapp for å verifisere, og en for å generere hash. Setter alignment til knappene og legger de også til i MainPane vår.

```

83     btHash.setOnAction(new EventHandler<ActionEvent>() { // This part does exactly the same as
84         @Override public void handle(ActionEvent e) { // No more comments for this one, look
85             Stage stage = new Stage();
86             File fileToWorkWith = new File(directory.getText());
87             String hash = hashFile.getText();
88             TextArea result = new TextArea();
89
90             result.setWrapText(true);
91             result.setStyle("-fx-text-fill: black");
92             result.setFont(Font.font("Times", 14));
93             result.setPrefWidth(599);
94             result.setPrefHeight(249);
95             result.setEditable(false);
96             ArrayList<String> toPrint = WhereAllTheMagicHappens.getAnswerForGUIhash(fileToWorkWith, hash);
97             if(toPrint.isEmpty()){
98                 result.appendText("Ingen resultat mottat \n");
99             }
100             else {
101                 for(int i = 0; i < toPrint.size(); i++){
102
103                     result.appendText((String) toPrint.get(i) + "\n");
104                 }
105             }
106             WhereAllTheMagicHappens.clearGUIResponse();
107             ScrollPane scrollPane = new ScrollPane(result);
108
109             Scene scene = new Scene(scrollPane, 600, 250);
110             stage.setTitle("Results Hashing Files");
111             stage.setScene(scene);
112             stage.show();
113         }
114     });
115

```

Figur 15 - btHash & btVer

Her ser du neste skritt på veien. Vi har to av disse – btHash og btVer. Det denne actionHandlern gjør er at når brukeren trykker på en knapp setter programmet fileToWorkWith som ny fil med navnet brukeren la inn (kan også være en dir). Vi setter også Stringen hash til filnavnet brukeren la inn der. Vi lager oss en ny stage som vi fyller med et tekstfelt (textarea nå, ikke textfield). Vi setter vidden, høyden, fontstil og størrelse. Vi stenger muligheten til brukeren for å kunne skrive inn i dette området også. Vi setter så dette inn i en scrollpane, da vi ønsker å kunne scrolle gjennom tilbakemeldingene vi får. Deretter setter vi scrollpane inn i vår nye stage og viser stagen vår.

Det neste vi gjør (dette er egentlig allerede gjort og lagt inn, men for forklaringens del ...) er å sende de 2 dataverdiene vi har inn til WhereAllTheMagicHappens, da enten for å lage hash eller for å verifisere hash. Alt etter hva brukeren har valgt. Returverdien er som nevnt tidligere en array med strings. Vi går gjennom listen og skriver ut hver setning (kan inneholde feilmelding, bekreftelse, avslag eller linje med fil hashet og hash).

Begge disse to metodene gjør akkurat det samme, med unntak av at btHash sender data inn til getAnswerForGUIhash. Og btVer sender data inn til getAnswerForGuiVer.

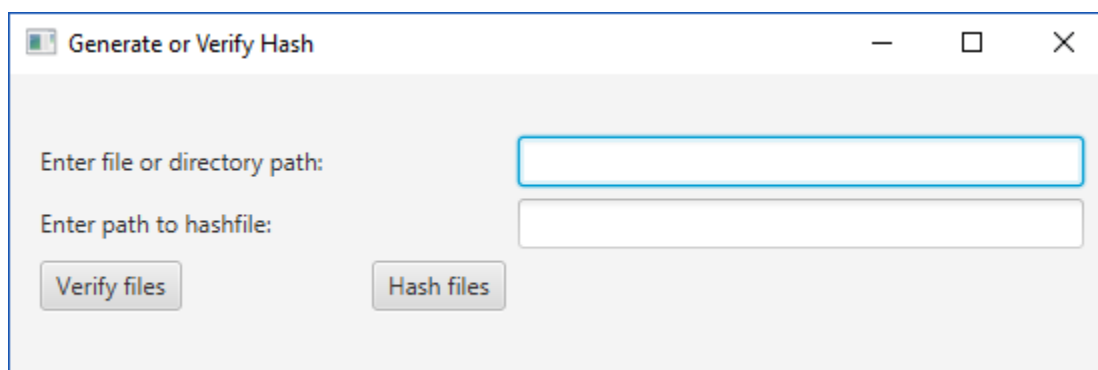
Siste 13 linjer med kode i denne filen er for å sette scenen for «hoved» scenen og vise denne. Legger derfor ikke inn noen ekstra forklaring eller bilde fra den kodesnutten. Se eventuelt i selve filen.

4 Hvordan bruke programmet

Kort bruksanvisning for de to forskjellige modusene du kan benytte mitt program i.

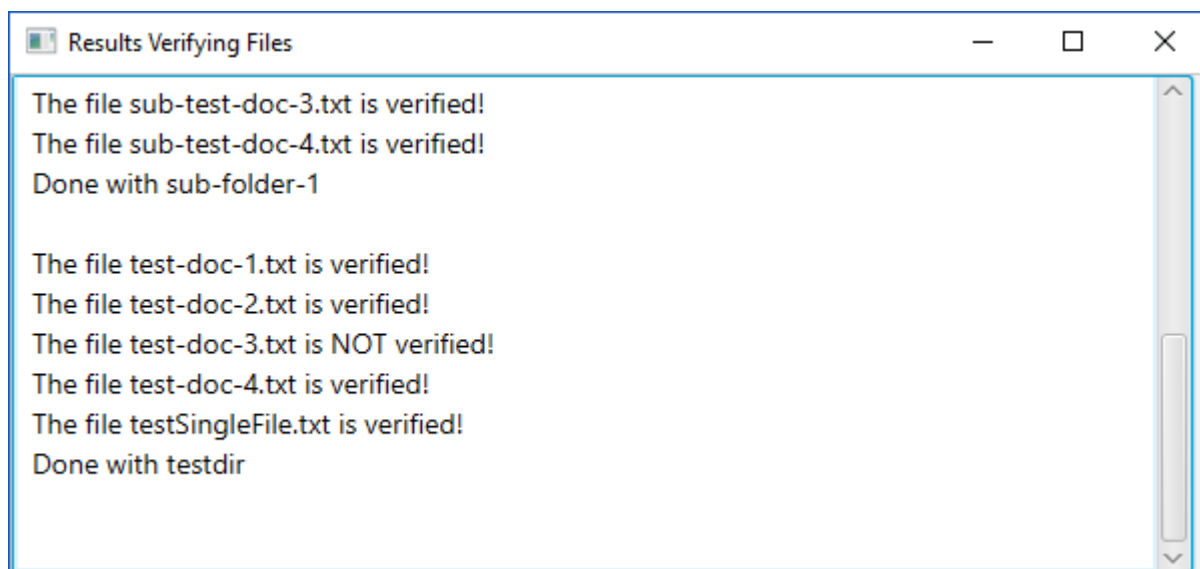
4.1 I GUI modus

Vi starter med GUI. Det er denne modusen jeg ville foretrukket, og den så enkel i bruk som man kan tenke seg.



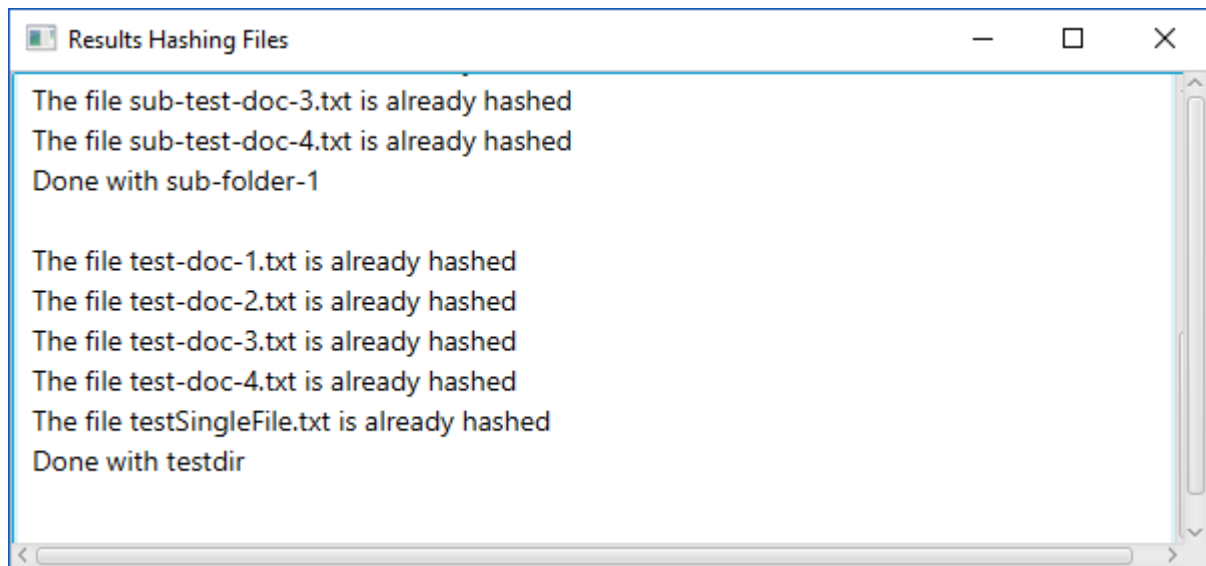
Figur 16 – TheGUI

Slik ser selve programmet ut når du kjører det fra skrivebordet. 2 felt som skal fylles ut, og en knapp alt etter hva brukeren ønsker skal gjøres.



Figur 17 - Verifiserer filer

Slik ser resultatet ut når brukeren verifiserer filer.



Figur 18 - Eksempel hashing, men filene er allerede hashet

Om filene i dette tilfellet ikke allerede hadde hatt en hashverdi i filen vi har bedt programmet skrive til ville vi fått svar med filnavn og hashverdi som er lagret. Se eksempel under.



Figur 19 - Eksempel hashing

MERK: Når du kjører i GUI kreves det ikke at du legger inn navn på filen du ønsker hashen skal lagres til. Om du ikke legger inn noe i det feltet vil programmet lete etter en fil ved navn TheHashList.txt, eller opprette en slik fil og lagre hash til. Alt etter om du verifiserer eller genererer.

Vær oppmerksom på at den leter kun i katalogen programmet kjører fra, og lagrer eventuelt TheHashList.txt dit også.

4.2 Via kommandolinje

For å kunne benytte programmet i kommandolinjen må du først kompilere koden. Du navigerer da fremt til mappen du har programmet liggende i, og kjører kommandoene:

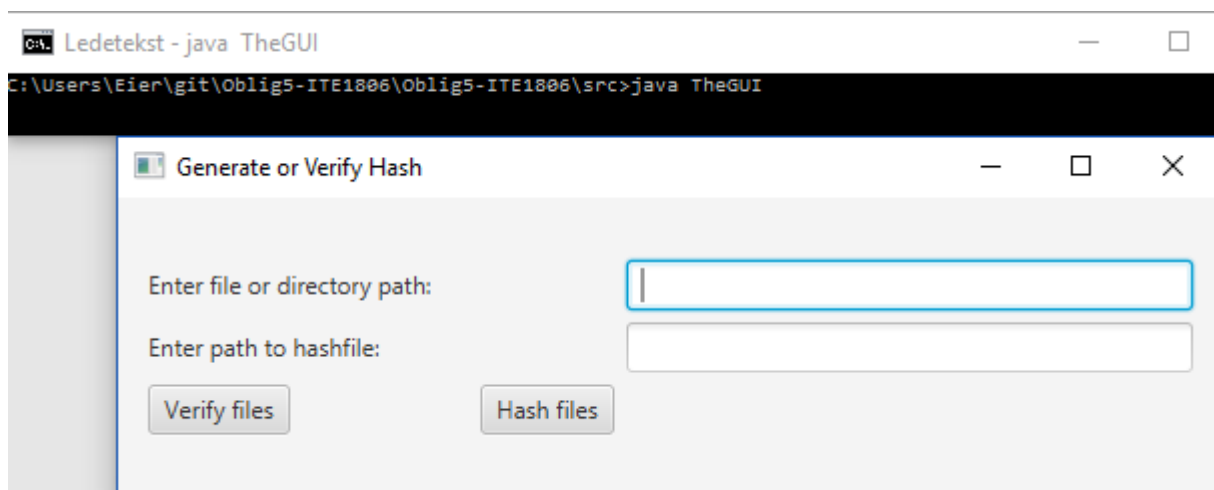
«**javac -cp cc110.jar; WhereAllTheMagicHappens.java**»

«**javac -cp cc110.jar; GVH.java**»

«**javac -cp cc110.jar; TheGUI.java**»

Cc110.jar er biblioteket som håndterer hash. Jeg har strippet bort ekstra filer, og forkortet navnet. Kun grunnet brukervennlighet. Fullt navn er commons-codec-1.10.jar

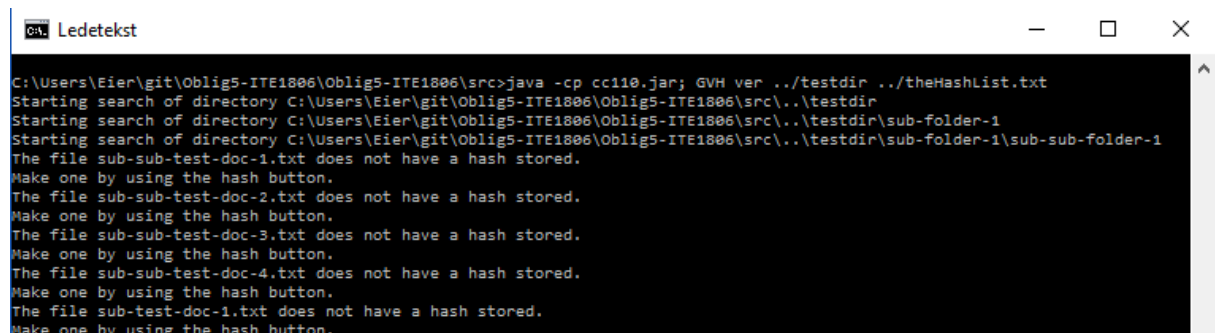
Når dette er gjort er programmet klart for bruk. Ønsker du å benytte programmet i GUI form kan du starte det med å kun skrive inn kommandoen (da fortsatt at du har navigert deg til src mappen du har javafilene i, at du bytter ut javac med java):



Figur 20 - Starte GUI fra kommandolinje

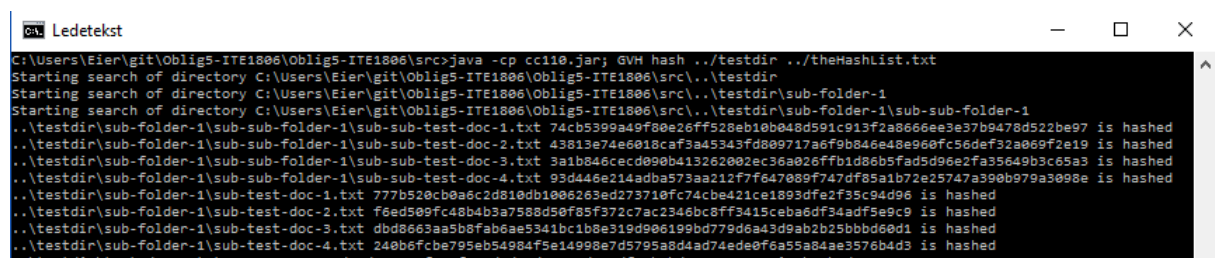
Utover dette har du 2 alternativer igjen. Du kan enten lage hash, eller verifisere filer. Uten videre utdypende forklaring vil de neste 2 skjermbildene vise hvordan det virker i praksis. Eneste som er verdt å merke seg er at du må legge til classpath for hver gang du skal kjøre programmet. Det gjøres ved «-cp cc110.jar;» og derfor er denne filen endret i navn og forkortet.

Se eksempelbilder på neste side..



```
C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src>java -cp cc110.jar; GVH ver ../testdir ../theHashList.txt
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir\sub-folder-1
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir\sub-folder-1\sub-sub-folder-1
The file sub-sub-test-doc-1.txt does not have a hash stored.
Make one by using the hash button.
The file sub-sub-test-doc-2.txt does not have a hash stored.
Make one by using the hash button.
The file sub-sub-test-doc-3.txt does not have a hash stored.
Make one by using the hash button.
The file sub-sub-test-doc-4.txt does not have a hash stored.
Make one by using the hash button.
The file sub-test-doc-1.txt does not have a hash stored.
Make one by using the hash button.
```

Figur 21 - Eksempel på verifisering i kommandolinje



```
C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src>java -cp cc110.jar; GVH hash ../testdir ../theHashList.txt
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir\sub-folder-1
Starting search of directory C:\Users\Eier\git\Oblig5-ITE1806\Oblig5-ITE1806\src\..\testdir\sub-folder-1\sub-sub-folder-1
..\testdir\sub-folder-1\sub-sub-folder-1\sub-sub-test-doc-1.txt 74cb5399a49f80e26ff528eb10b048d591c913f2a8666ee3e37b9478d522be97 is hashed
..\testdir\sub-folder-1\sub-sub-folder-1\sub-sub-test-doc-2.txt 43813e74e6018caf3a45343fd809717a6f9b846e48e960fc56def32a069f2e19 is hashed
..\testdir\sub-folder-1\sub-sub-folder-1\sub-sub-test-doc-3.txt 3a1b846cecd090b413262002ec36a026ffb1d86b5fad5d96e2fa35649b3c65a3 is hashed
..\testdir\sub-folder-1\sub-sub-folder-1\sub-sub-test-doc-4.txt 93d446e214adba573aa212f7f647089f747df85a1b72e25747a390b979a3098e is hashed
..\testdir\sub-folder-1\sub-test-doc-1.txt 777b520cb0a6c2d810db1006263ed273710fc74cbe421ce1893dfe2f35c94d96 is hashed
..\testdir\sub-folder-1\sub-test-doc-2.txt f6ed509fc40b4b3a7588d50f85f372c7ac2346bc8ff3415ceba6df34adff5e9c9 is hashed
..\testdir\sub-folder-1\sub-test-doc-3.txt dbd8663aa5b8fab6ae5341bc1b8e319d906199bd779d6a43d9ab2b25bbbd60d1 is hashed
..\testdir\sub-folder-1\sub-test-doc-4.txt 240b6fcbe795eb54984f5e14998e7d5795a8d4ad74ede0f6a55a84ae3576b4d3 is hashed
..\testdir\test-doc-1.txt a1c2442453467d72d511366fc38fec7dcd6d408c5d35cdf73b0bdc718a32237 is hashed
```

Figur 22 - Eksempel på hashing i kommandolinje

5 Testkjøring

Under vises en tabell med verdier og resultat av testkjøring. Filer markert med en * har fått endringen «I AM CHANGED AFTER HASHING» er lagt til etter at hashen er generert.

Ved en testkjøring av filer som er levert med rapporten skal du få samme resultat.

Tabell 1 - Resultat av testkjøring av programmet.

FILNAVN	HASH	VER
TESTSINGLEFILE	a3bbd1daf2464e23bc19b427272b805c5838614bbca620ca f22ea32491472042	Ja
TEST-DOC-1	e1c24424524c7d72d511366fc30fec7dedc6d408c5d35cdf2 7b0bdc718a322a7	Ja
TEST-DOC-2	874d036b28dc8d7e8258d74ae7efe3a0a7224ebd892ae5b4 97b677135943412e	Ja
*TEST-DOC-3	3b04c4d315003fc272b995bef291b5e0c7a2f595f8a6221ea cb7ea4dca80d532	Nei
TEST-DOC-4	37b85133fee4871daf2d2db37d62acfd41b445b521bf816 67d4a90df10d993a	Ja
SUB- TEST-DOC-1	777b520cb0a6c2d810db1006263ed273710fc74cbe421ce1 893dfe2f35c94d96	Ja
*SUB- TEST-DOC-2	3b0dc6693b68378b0fb3839796579ab855b5f0a97b533c5b e1a78098c4b22b49	Nei
SUB- TEST-DOC-3	dbd8663aa5b8fab6ae5341bc1b8e319d906199bd779d6a43 d9ab2b25bbbd60d1	Ja
SUB- TEST-DOC-4	240b6fcbe795eb54984f5e14998e7d5795a8d4ad74ede0f6a 55a84ae3576b4d3	Ja
SUB- SUB- TEST-DOC-1	74cb5399a49f80e26ff528eb10b048d591c913f2a8666ee3e 37b9478d522be97	Ja
*SUB- SUB- TEST-DOC-2	66514d1a304b6b0a767aa3e2f6fdf4eda95ff63c13b735b65 4ffc085142401b7	Nei
SUB- SUB- TEST-DOC-3	3a1b846cecd090b413262002ec36a026ffb1d86b5fad5d96e 2fa35649b3c65a3	Ja
SUB- SUB- TEST-DOC-4	93d446e214adba573aa212f7f647089f747df85a1b72e2574 7a390b979a3098e	Ja

6 Problemer og utfordringer

Har møtt på svært lite problemer i denne oppgaven, og de største problemene jeg har hatt er å bli enig med meg selv i designvalg. Jeg ønsket at det skulle være så enkelt som mulig å bruke og endte derfor med å gå primært for GUI løsning. Hvor kommandolinjen er i all hovedsak en «bonus» eller nødløsning.

Men jeg møtte litt problemer når det gjelder å kjøre programmet i kommandolinjen. Jeg måtte google litt for å finne ut av -cp⁴ kommandoen, og jeg har ikke klart å finne en måte å unngå at brukerne vil være nødt til å legge inn denne ved hver kjøring. Det problemet gjelder da ikke for GUI versjonen.

Jeg måtte også oppsøke stackOverflow for hjelp til hvordan lage en ny stage⁵ ved en actionEvent. Jeg har ikke stilt spørsmål eller snakket med noen, men fant spørsmål andre har stilt rundt det samme.

Alt i alt har oppgaven gått ganske greit og selve javaboken⁶ var kilden til stort sett alt jeg trengte.

Men kan legge til at programmet kan helt klart bli litt mer strømlinjeformet, og en del kode gjentar seg selv unødvendig. Ikke at jeg vil betegne det som galt, men kunne vært kortet ned litt, og med litt flere felles metoder. Med andre ord, hadde jeg vært litt smartere i fremgangsmåten kunne koden vært litt mer effektiv.

Det er også en liten bug jeg ikke klarte å bli kvitt. Du får startet opp GUI versjonen fra kommandolinjen uten å legge til classpath (se figur 20), men programmet feiler når du prøver å kjøre det. Kommer litt tilbake til det at jeg ikke fant ut hvordan jeg får inkludert biblioteket fra common-codec i på noe vis fra start.

Men jeg er alt i alt fornøyd med resultatet.

⁴ <http://stackoverflow.com/questions/17529717/how-to-run-java-from-cmd-with-apache-commons-libraries>

⁵ <http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm>

⁶ Introduction to Java Programming Comprehensive Version 10th-Edition by Y. Daniel Liang

Referanseliste

Komplett liste over alle plasser jeg har funnet hjelp når jeg har stått fast eller hatt spørsmål jeg har lurt på. Alle disse kildene er også referert til i fotnote på sidene de er relevante til.

- <http://stackoverflow.com/questions/20963700/how-to-open-another-window-in-javafx-2>
- <http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm>
- <http://kark.hin.no/opsys/tanenbaum/java/ExecutableFiles.java>
- http://kark.hin.no/sikkerhet/filehash/oblig5_tips.html
- <http://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/digest/DigestUtils.html>
- <http://stackoverflow.com/questions/17529717/how-to-run-java-from-cmd-with-apache-commons-libraries>
- Introduction to Java Programming Comprehensive Version 10th-Edition by Y. Daniel Liang

Vedlegg

Alle vedlegg ligger som egne filer, og er som følger:

- theHashList.txt
- src (mappe)
 - cc110.jar
 - WhereAllTheMagicHappens.java
 - GVH.java
 - TheGUI.java
- Testdir – dette er en mappe med totalt 13 tekstfiler i. Benyttet til testing av programmet.
- GVH.jar (Programmet i skrivebordsform)