

Inhaltsverzeichnis

Aufgabe zu nebenläufiger Programmierung - Hash it!	1
Beschreibung Ausgangsprogramm	1
Aufgabe.....	1
GIT Link	2
Tasks	2
Analyse.....	3
Herangehensweise	4
Ergebnis-Klasse	4
Umbau der Klasse Hashfinder.....	4
Umbau der Klasse „Main“	5
Ergebnis	6

Aufgabe zu nebenläufiger Programmierung - Hash it!

Beschreibung Ausgangsprogramm

In diesem kleinen Script wird in der main Methode (App.java) `findHash(String data, String difficulty)` aufgerufen.

`findHash` hängt an `String data` einen `nonce` (Zahl, die in jedem Durchlauf hochgezählt wird) an und generiert einen Hashcode. Fängt dieser Hashcode mit den Chars an, die in `String difficulty` übergeben worden sind, ist die Aufgabe erledigt.

Oder kurz gesagt: Das Programm muss umso mehr Arbeit verrichten, umso mehr Nullen du in `String difficulty` übergibst. Nicht anders funktioniert im Prinzip das Mining bei Bitcoin.

Aufgabe

Nutze das bisher angesammelte Wissen, um die main Methode per Multithreading effizienter zu machen. Es sollen mehrere Threads gemeinsam den gewünschten Nonce suchen.

Die ersten Punkte gibt es, wenn du es überhaupt geschafft hast, `findHash(String data, String difficulty)` parallel auszuführen.

Die nächsten Punkte gibt es, wenn du einen Weg findest, dass die parallel gestarteten `findHash(String data, String difficulty)` Aufrufe nicht die selbe Arbeit verrichten (das wäre ja unsinnig). Bis zu diesem Punkt kann alles in der Main Methode gemacht werden.

Zusatzpunkte gibt es, wenn du es auch noch schaffst, dass alle Threads abbrechen, wenn ein Thread den gesuchten Hash gefunden hat. Hierfür wirst du auch Änderungen in der Klasse `HashFinder` durchführen müssen. Bis zu diesem Punkt wurde alle notwendige Theorie bereits umfassend behandelt.

Noch mehr Punkte kann man sich holen, wenn für das Multithreading Higher Level Bibliotheken wie Executor Service verwendet werden.

GIT Link

<https://github.com/Doc-Gonzo/HashFinder.git>

Tasks

- findHash entsprechend ändern
- mit verschiedenen Parametern in parallelen Tasks laufen lassen
- erfolgreicher Thread muss anderen benachrichtigen (beenden)

Analyse

Die Variable „nonce“ vom Typ „long“ steuert augenscheinlich die Generierung des Hashes. Sie dient als Startpunkt.

```
7 System.out.printf("Nonce:" + nonce);  
8 System.out.printf("%s - %s %n", Thread.currentThread().getName(), hash);
```

Wird das Script mehrmals aufgerufen, wird „Nonce“ Nr. 9172 immer den selben Zielhash erzeugen.

```
Nonce:9165main - 97bc1711780d1b57cab03d905d5307f985d682c658503a9a779c95f66982d8c5  
Nonce:9166main - 31921b12cc72cfd3d3a9c321986420a01c2760b00ef3df81620377fa452da3cb  
Nonce:9167main - 6f94b9538aab44ffb7769679a11537bd725264748c202fdae48b312400ebad63  
Nonce:9168main - 27a00b53736c4168c264ed0a8145289a2bc7437bc80bec8045b3a5bf73c8a2dd  
Nonce:9169main - 1341f1a1d9ce8667c11a3d81897ce0d1b1ae747453f8c1e14d2c49c4f0108357  
Nonce:9170main - f2ee882ec39ca899e46a0c650ccb6fc9ccce2fa1633ab0decc6c0cffacbb21  
Nonce:9171main - a0574bc8ca50ed535ee0e43d37cd4afef660002ffb9879986227a68f33723aaa  
Nonce:9172main - 00001f2e9f8f74117b4178eb04b368c807f906ae2a07bece562266cbc9adff3c
```

Um „findHash“ sinnvoll parallel einsetzen zu können, muss die Methode mit verschiedenen Startpunkten aufgerufen werden können.

Wie finden wir aber einen Range für „nonce“ heraus, welchen die einzelnen Threads abarbeiten sollen? Dazu rufen wir das Script mit den original Startparametern auf, und sehen das knapp über 9000 Hashes generiert wurden.

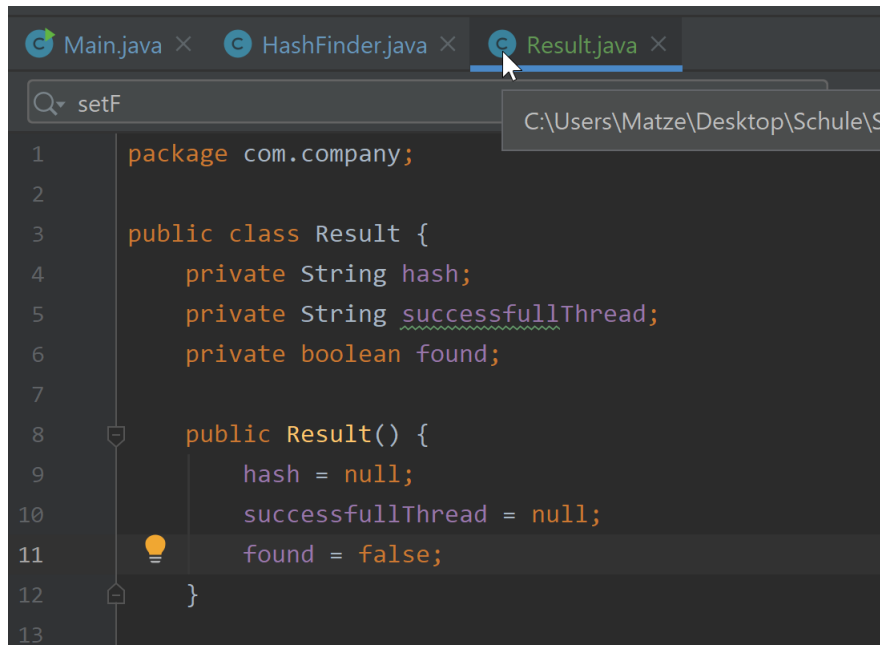
```
Nonce:9172main - 00001f2e9f8f74117b4178eb04b368c807f906ae2a07bece562266cbc9adff3c  
  
Process finished with exit code 0
```

5000 Hashes pro Thread scheint bei diesen Einstellungen ein guter Startwert zu sein. Zwei Threads müssten die Aufgabe gemeinsam also mehr als doppelt so schnell bewältigen können.

Herangehensweise

Ergebnis-Klasse

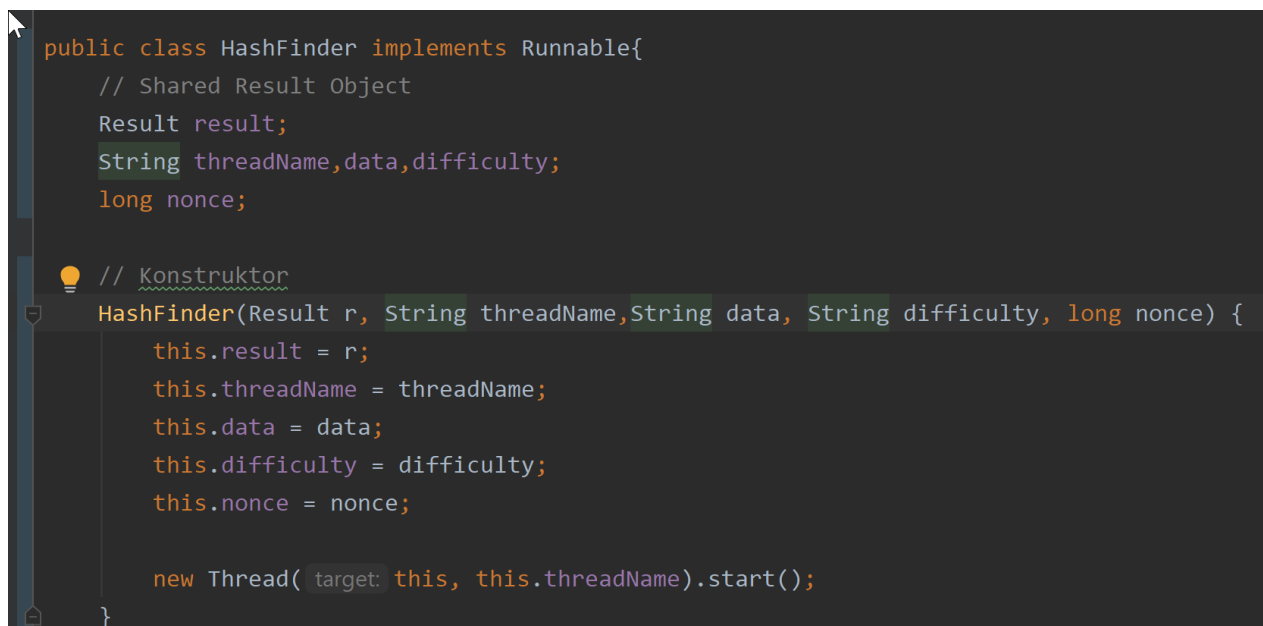
Damit Threads kommunizieren können, wird ein geteiltes „Result“-Objekt erzeugt. Die Klasse beinhaltet folgende Variablen:



```
1 package com.company;
2
3 public class Result {
4     private String hash;
5     private String successfullThread;
6     private boolean found;
7
8     public Result() {
9         hash = null;
10        successfullThread = null;
11        found = false;
12    }
13
```

Umbau der Klasse Hashfinder

Die Klasse „Hashfinder“ wird umgebaut, und implementiert nun „Runnable“. Die Klasse selbst bekommt alle Variablen, die vorher der Methode mitgegeben wurden, sowie ein Result-Objekt. Der Wert „nonce“ wird ebenfalls im Konstruktor mitgegeben.



```
public class HashFinder implements Runnable{
    // Shared Result Object
    Result result;
    String threadName,data,difficulty;
    long nonce;

    // Konstruktor
    HashFinder(Result r, String threadName,String data, String difficulty, long nonce) {
        this.result = r;
        this.threadName = threadName;
        this.data = data;
        this.difficulty = difficulty;
        this.nonce = nonce;

        new Thread( target: this, this.threadName).start();
    }
}
```

Die Logik der Methode wird in den run()-Block gepackt:

```
// Run Methode checkt ob Hash im shared objekt gefunden
public void run() {
    while (!result.isFound()) {
        do {
            String strToHash = data+nonce;
            String hash = calculateHash(strToHash);
            System.out.printf("%s - %s %n",Thread.currentThread().getName(),hash,"Nonce:" , nonce);
            if(hash.startsWith(difficulty)){
                result.setFound(true);
                System.out.printf("Hash gefunden: %s - %s %n",Thread.currentThread().getName(),hash,"Nonce:" + nonce);
            }
            nonce++;
        } while (!result.isFound() && nonce < Long.MAX_VALUE);
    }
}
```

Hier wird nun auf „isFound()“ des Result-Objektes geprüft, statt auf die MethodenvARIABLE der Klasse selbst.

```
// Run Methode checkt ob Hash im shared objekt gefunden
public void run() {
    while (!result.isFound()) {
        do {
```

Umbau der Klasse „Main“

Zuerst wird ein Result-Objekt generiert, auf das beide Threads zugreifen.

Anschließend werden zwei Threads gestartet, mit „nonce“-Werten von 0 und 5000:

```
public class Main {

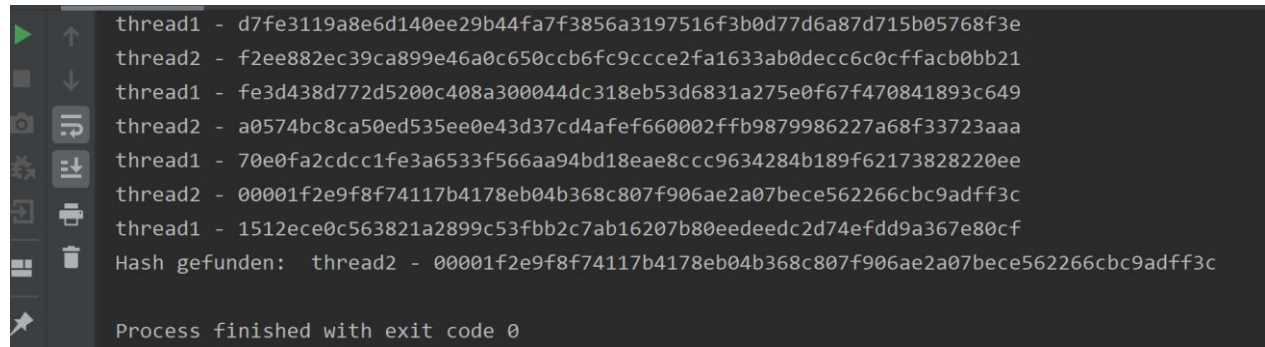
    public static void main(String[] args) {
        // Ergebnisobjekt, auf das alle Threads zugreifen können
        Result result = new Result();

        System.out.println("Hello World!");
        // HashFinder.findHash("Hello","0000");

        HashFinder finder1 = new HashFinder(result, threadName: "thread1", data: "Hello", difficulty: "0000", nonce: 0);
        HashFinder finder2 = new HashFinder(result, threadName: "thread2", data: "Hello", difficulty: "0000", nonce: 5000);
    }
}
```

Ergebnis

Thread Nr.2 fand ein Ergebnis, das Programm endet. Eine letzte Iteration von Thread1 wurde allerdings noch abgearbeitet, bevor das Programm beendet.

A screenshot of a debugger's thread list and console output. The thread list on the left shows two threads, thread1 and thread2, with their respective memory addresses. The console output on the right shows the execution of these threads, with thread2 finding a hash and the program finishing with exit code 0.

```
thread1 - d7fe3119a8e6d140ee29b44fa7f3856a3197516f3b0d77d6a87d715b05768f3e
thread2 - f2ee882ec39ca899e46a0c650ccb6fc9ccce2fa1633ab0decc6c0cfffacbb21
thread1 - fe3d438d772d5200c408a300044dc318eb53d6831a275e0f67f470841893c649
thread2 - a0574bc8ca50ed535ee0e43d37cd4afef660002ffb9879986227a68f33723aaa
thread1 - 70e0fa2cdcc1fe3a6533f566aa94bd18eae8ccc9634284b189f62173828220ee
thread2 - 00001f2e9f8f74117b4178eb04b368c807f906ae2a07bece562266cbc9adff3c
thread1 - 1512ece0c563821a2899c53fbb2c7ab16207b80eedeecd2d74efdd9a367e80cf
Hash gefunden: thread2 - 00001f2e9f8f74117b4178eb04b368c807f906ae2a07bece562266cbc9adff3c

Process finished with exit code 0
```