Password Cracker Simulation 50 Punkte

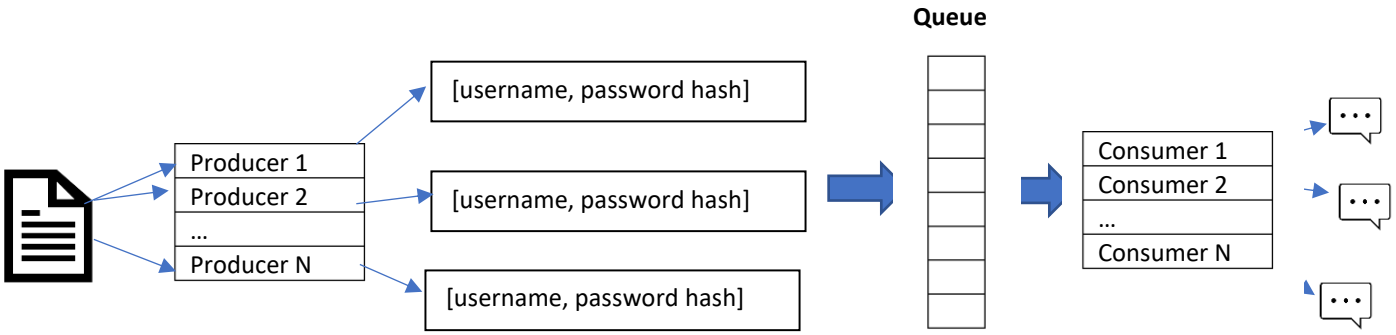
Einleitung

Sie sollen für eine Security Firma eine Passwort-Cracker Software entwickeln. Die Anforderungen an das System sind vor allem eine hohe Skalierbarkeit (d.h. die Nutzung mehrerer Threads) für die Ermittlung des Passwortes. Ebenfalls sollen verschiedene Arten des Passwort-Crackings implementiert werden können. Die Software soll deshalb leicht erweiterbar sein, um neue Strategien leicht implementieren zu können, ohne die bestehende Software verändern zu müssen. Die Ergebnisse sollen ebenfalls, leicht erweiterbar, verschiedenen Empfängern (zum Beispiel: Standard-Output, Log-File, Socket) zur Verfügung gestellt werden können.

Ihre Firma entscheidet sich dafür, die Anwendung nach dem Producer-Consumer Pattern zu implementieren. Sie haben die Aufgabe, eine Beispiel-Implementation umzusetzen.

Ablauf der Beispielapplikation

- Sie erhalten eine Textdatei mit Passwörtern (1 Passwort pro Zeile)
- Mehrere Threads (PasswordProducer) erzeugen zufällig einen Eintrag bestehend aus `username` und `passwordHash` und fügen ihn zur zentralen Queue hinzu.
- Mehrere Threads (PasswordConsumer) holen sich Einträge von der zentralen Queue und versuchen das Passwort zu erraten (mit Hilfe einer implementierten Strategie). Wurde das Passwort erraten / nicht erraten, wird das Ergebnis veröffentlicht.



Implementation

Es ist eine Java-Standard Applikation mit Konsolenausgabe zu erstellen.

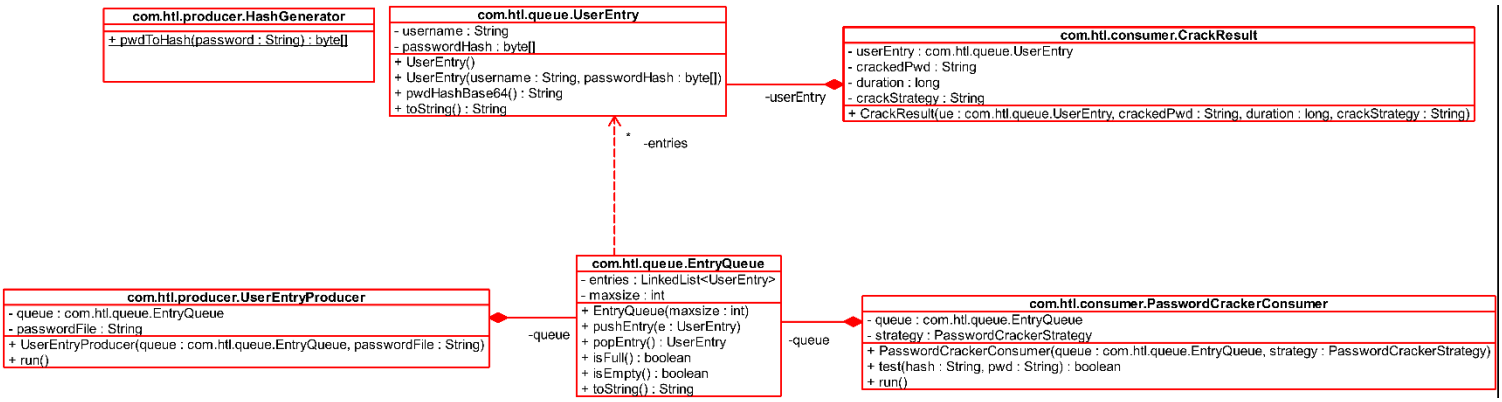
Die Implementation gliedert sich in mehrere Teilschritte:

1. Implementation des Producer-Consumer
2. Implementation der Strategien für die Consumer (Passwort-Cracker)
3. Testprogramm
4. Erweiterung der Ergebnispräsentation für die Consumer

Anmerkung: Erweiterung der Klassen, ableiten von bestehenden Klassen der `java threading api` sind ausdrücklich erlaubt, erwünscht und teilweise notwendig.

ad 1) Producer-Consumer Implementation 22 Punkte

Klassendiagramm



Klassenbeschreibung

Klasse `UserEntry`
- username: Name des Benutzers
- passwordHash: Hash des Passwortes
+ UserEntry(…): Initialisiert das Objekt mit den `gegebenen` Werten
+ pwdHashBase64(): passwordHash als Base64 kodierter String
+ toString(): Stringrepräsentation in der Form `[username]:␣[passwordHashBase64]`

Klasse `HashGenerator`
+ pwdToHash(password:String): Returns the SHA256 hash (as byte array) of the given string.

Klasse `EntryQueue`
- entries: Liste der UserEntry einträge in der Warteschlange
- maxsize: Die maximale Größe der Warteschlange
+ pushEntry(UserEntry): fügt ein neues Element hinzu. Ist die Warteschlange voll, wartet der hinzufügende Thread, bis wieder Platz in der Warteschlange ist.
+ popEntry(UserEntry): entfernt ein Element aus der Warteschlange. Ist die Warteschlange leer, wartet der Thread, bis wieder Elemente in der Warteschlange zur Verfügung stehen.
+ isFull(): true, wenn die Warteschlange voll ist.
+ isEmpty(): true, wenn die Warteschlange leer ist.

Klasse `UserEntryProducer`
- queue: Die Warteschlange, für die die Einträge erzeugt werden
- passwordFile: Die Passwörter-Datei
+ run(): Erzeugt jede Sekunde einen neuen Eintrag für die Warteschlange nach folgendem Vorgehen:
  username: `"user"[MillisecondTime]`
  passwordHash: SHA256 Hash eines zufälligen Passwortes aus der Datei

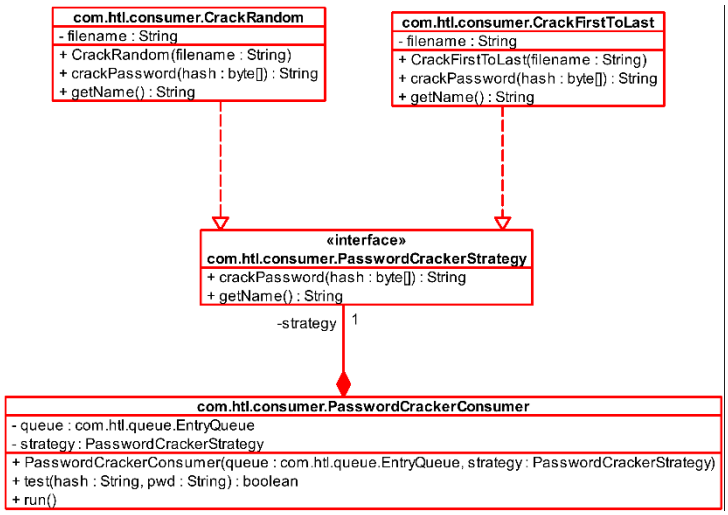
Klasse `CrackResult`
- userEntry: Das UserEntry Objekt, das geknackt wurde
- crackedPwd: Das Passwort im Klartext
- duration: Die Dauer des Passwortknackens in ms
- crackStrategy: Name der verwendeten Strategie

Klasse `PasswordCrackerConsumer`
- queue: Die Warteschlange, für die die Einträge erzeugt werden
- strategy: Objekt zum Knacken des Passwortes
+ run(): Holt sich laufend Einträge von der Warteschlange und verwendet das `strategy` Objekt um das Passwort zu knacken. Ausgabe (auf Konsole oder bei Erweiterung mittels Observer) des Ergebnisses wie folgt:

`[username]:␣[passwordHashBase64]␣=␣'[password]'␣([duration]␣-␣[strategy])`

ad 2) Strategien — 10 Punkte

Klassendiagramm



Klassenbeschreibung

Klasse `PasswordCrackerConsumer`
siehe ad 1)

Interface `PasswordCrackerStrategy`
+ crackPassword(): Für einen gegebenen Passworthash wird das Klartextpasswort zurückgegeben
+ getName(): Der Name der Strategie

Klasse `CrackRandom`
- filename: Der Dateiname der Passwortdatei
+ crackPassword(hash): Holt sich solange zufällige Passwörter aus der Datei, berechnet den Passworthash und vergleicht ihn mit dem gegebenen Hash bis beide Hashes identisch sind. Danach wird das Klartextpasswort zurückgegeben.

Klasse `CrackFirstToLast`
- filename: Der Dateiname der Passwortdatei
+ crackPassword(hash): Holt sich alle Passwörter der Reihe nach aus der Datei, berechnet jeweils den Passworthash und vergleicht ihn mit dem gegebenen Hash. Wenn beide Hashes identisch sind, wird das Klartextpasswort zurückgegeben.

ad 3) Testprogramm — 8 Punkte

Schreiben Sie ein Test-Programm, das die Datei „`rockyou_small.txt`" verwendet, um ihren Passwort Cracker zu testen. Anmerkung: Verwenden Sie unbedingt einen relativen Pfad zur Datei!

Erstellen Sie dafür folgendes:

- Eine `EntryQueue` mit maximal 10 Elementen
- 2 `UserEntryProducer` Threads
- 4 `PasswordCrackerConsumer` Threads wobei hier 2 Threads mit Hilfe der Strategie `CrackFirstToLast` und 2 Threads mit Hilfe der Strategie `CrackRandom` die Passwörter knacken sollen.

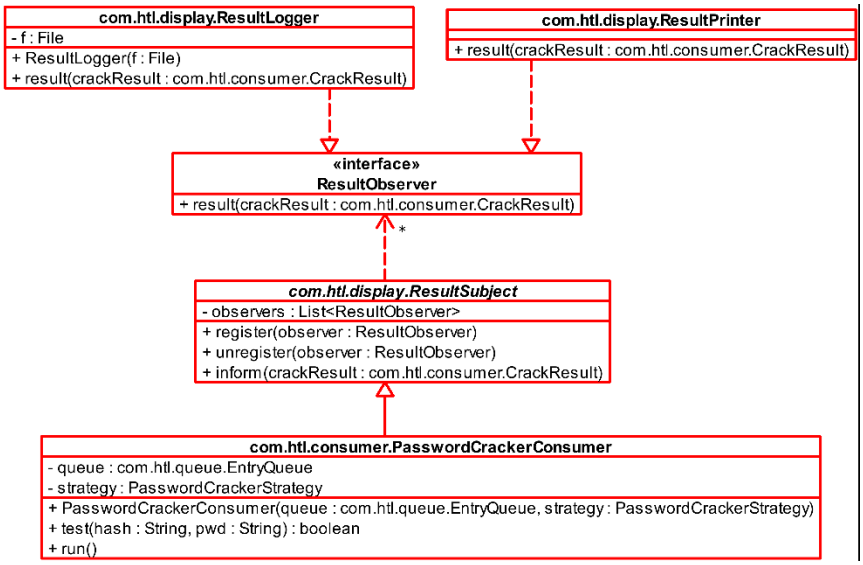
Die Ausgabe sollte dann so aussehen:

```
user1611833445356: 'eQfPYDuLZDOC8EnlaVT9ZLfK3IkrIrvaj39kUZIex+0=' = '061883' (220 - CrackFirstToLast)
user1611833445356: 'sRbP784ccWCGqZxsNcwbebhu9JwYz+Fu3IrJlMzWdFA=' = 'pisces07' (304 - CrackRandom)
user1611833446382: '33NixF6llVY2TmiECbjD7lbpMv265yN9OkIyeniUsiU=' = 'Malachi' (64 - CrackRandom)
user1611833446390: 'siH30VbakFeMG4N/7DTijKnlhfNnT8QpypmFa6aIaZg=' = '10031988' (70 - CrackRandom)
user1611833447394: 'O5t6FZkx/J8ubh5NaQpDddTnIwOaJVgf3VJQWNl2rPM=' = '081583' (43 - CrackRandom)
user1611833447394: 'NMqboDTBUemxqEsIRhtgLwZQdt99lrFMm6dGA9B7EE4=' = 'nikol2' (71 - CrackFirstToLast)
```

ad 4) Erweiterung Ergebnispräsentation — 8 Punkte

Ersetzen Sie die einfache Ausgabe des `PasswortCrackerConsumer` durch folgende, erweiterbare, Ergebnispräsentation.

Klassendiagramm



Klassenbeschreibung

Klasse `PasswordCrackerConsumer`
siehe ad 1)

Klasse `ResultSubject`
- observers: Liste der Beobachter
+ register(): Hinzufügen eines Beobachters
+ unregister(): Entfernen eines Beobachters
+ inform(): Informiert alle Beobachter über eine neues Ergebnis

Interface `ResultObserver`
+ result(): Wird bei einem neuen Ergebnis aufgerufen

Klasse `ResultLogger`
- f: Logdatei der Ergebnisse
+ result(): Für jedes Ergebnis wird eine neue Zeile in folgender Form an die Datei angefügt:

```
[ISO Timestamp];[passwordHashBase64];[password];[duration];[strategy name]
```

Klasse `ResultPrinter`
+ result(): Für jedes Ergebnis wird eine neue Zeile auf den Standard-Output in folgender Form geschrieben:

```
[username]:␣[passwordHashBase64]␣=␣'[password]'␣([duration]␣-␣[strategy])
```

ad 5) Erweiterung Testprogramm — 2 Punkte

Erweitern Sie das Testprogramm so, dass alle `PasswordCrackerConsumer` Threads ihre Ergebnisse sowohl an die Konsole als auch an die Datei weiterleiten.