
# Document pentru Proiectarea Software a Soluției **Aranea**

Apostolescu Ștefan
Băjan Ionuț-Mihăiță
Brumă Ionuț-Cosmin
Iosif George-Andrei
Stanciu Răzvan-Daniel

30/11/2020

# Tabelă de Conținut

**1 Detalii despre Document** **4**
1.1 Scop . . . . . 4
1.2 Conținut . . . . . 4

**2 Proiectare a Datelor** **5**
2.1 Formate ale Fișierelor . . . . . 5
2.2 Structuri de Date . . . . . 7

**3 Proiectare a Arhitecturii** **8**
3.1 Diagramă de Clase . . . . . 8
3.2 Descriere a Claselor . . . . . 9

**4 Modelare a Interfeței cu Utilizatorul** **18**

**5 Elemente de Testare** **19**

# Listă de Imagini

1 Diagramă de clase . . . . . 8

# Listă de Tabele

1 Tipuri de fișiere cu formate speciale . . . . . 5
2 Structuri de date globale . . . . . 7
3 Descriere a clasei `Logger` . . . . . 9
4 Descriere a clasei `Exception` . . . . . 10
5 Descriere a clasei `FolderParser` . . . . . 11
6 Descriere a clasei `ExtensionFinder` . . . . . 12
7 Descriere a clasei `PatternFinder` . . . . . 13
8 Descriere a clasei `SitemapGenerator` . . . . . 14
9 Descriere a clasei `Sieve` . . . . . 15
10 Descriere a clasei `Crawler` . . . . . 16
11 Descriere a clasei `Aranea` . . . . . 17

# Listă de Exemple

1 Fișier cu referințe către website-uri . . . . . 5
2 Fișier de configurare . . . . . 6
3 Extras dintr-un fișier de jurnalizare . . . . . 6

# 1 Detalii despre Document

## 1.1 Scop

Scopul acestui document este de a descrie modul în care va fi implementat sistemul Aranea, plecând de la cerințele specificate în documentul "*Specificarea Cerințelor Software*". Pe lângă aceasta, documentul de față ajută programatorii acestei echipe să își creeze o imagine de ansamblu asupra proiectului, lucru care favorizează alinierea dintre funcționalitățile finale și nevoile beneficiarilor.

## 1.2 Conținut

Documentul este împărțit în patru capitole. Primul prezintă maniera în care datele concrete, reale, sunt încapsulate în structuri de date în cadrul implementării software. Al doilea capitol oferă o descriere a arhitecturii proiectului, plecând de la enumerarea componentelor și ajungând până la modul în care acestea comunică între ele pentru a asigura o funcționare corectă. Al treilea descrie interfața cu utilizatorul, în timp ce ultimul prezintă cum va fi efectuată testarea soluției.

# 2 Proiectare a Datelor

## 2.1 Formate ale Fișierelor

O serie de fișiere cu un format special, prezentate în tabelul următor, sunt necesare execuției programului.

| Notație | Descriere | Tip |
|---|---|---|
| `URLS_FILE` | fișier cu referințe către website-uri ce trebuiesc descărcate | Intrare |
| `CONFIG_FILE` | fișier de configurare | Intrare |
| `LOG_FILE` | fișier de jurnalizare | Ieșire |

Tabel 1: Tipuri de fișiere cu formate speciale

Primul tip de fișier de intrare, `URLS_FILE`, conține referințe către acele website-uri ce trebuiesc descărcate de către Aranea. Un singur URL complet (prefixat și cu protocolul aferent, `http://` pentru HTTP și `https://` pentru HTTPS) trebuie să apară pe fiecare linie a unui astfel de fișier. Separarea între linii se face cu `\n` (`0x0A` în ASCII).

```
http://apache.org/
https://www.sts.ro/
```

Exemplu 1: Fișier cu referințe către website-uri

Fișierele de tip `CONFIG_FILE` sunt, de asemenea, fișiere de intrare. Scopul lor este de a seta configurația inițială a soluției prin intermediul anumitor chei:

- `download_dir`, șir de caractere pentru directorul de descărcare
- `log_file`, șir de caractere pentru fișierul de jurnal
- `log_level`, întreg pentru prioritatea minimă a unui eveniment pentru a fi jurnalizat *(opțional, implicit `0`)*
- `is_sitemap_generated`, boolean care indică dacă se vor genera hărți pentru website-urile descărcate *(opțional, implicit `true`)*
- `max_threads`, întreg pentru numărul maxim de fire de execuție *(opțional, implicit `1000`)*
- `delay`, numărul de secunde între două cereri consecutive către un server web țintă *(opțional, implicit `1`)*
- `allowed_extensions`, șir de caractere pentru extensii ale fișierelor ce vor fi descărcate, separate prin virgulă *(opțional, implicit `*`)*
- `allowed_max_size`, întreg pentru dimensiunea maximă, în octeți, avută de un fișier ce va fi descărcat *(opțional, implicit `1000000000`)*

- `allowed_pattern`, șir de caractere pentru un șablon Regex ce trebuie să se regăsească într-un fișier pentru a fi descărcat *(opțional, implicit `""`)*
- `skip_robotsdottxt_files`, boolean care indică dacă fișierele menționate în `robots.txt` nu vor fi descărcate *(opțional, implicit `true`)*.

Conform cu formatul standard al unui fișier de proprietăți, pe care fișierele de configurare îl respectă, fiecărei chei îi corespunde o valoare. De menționat ar fi că setarea anumitor chei este obligatorie și că toate cheile au o valoare implicită asignată, ce va fi folosită în cazul omiterii cheii în fișierul de configurare. Perechile chei-valoare trebuiesc separate între ele prin `\n` (`0x0A` în ASCII).

```
download_dir=/home/aranea/downloads
log_file=/home/aranea/logs
delay=0
skip_robotsdottxt_files=false
```

Exemplu 2: Fișier de configurare

Cât despre fișierele de jurnalizare, ele sunt produse în mod automat de către Aranea. Conțin mai multe linii separate de `\n` (`0x0A` în ASCII), pe fiecare linie aflându-se o marcare a unui eveniment ce s-a petrecut de-a lungul execuției. Marcările respectă formatul *`"[ZZ.LL.AA OO:MM:SS] TIP_EVENIMENT: MESAJ"`*, unde:

- `ZZ` este identificatorul numeric, de două cifre, al zilei
- `LL` este identificatorul numeric, de două cifre, al lunii
- `ZZ` este identificatorul numeric, de două cifre, al anului
- `OO` este identificatorul numeric, de două cifre, al orei
- `MM` este identificatorul numeric, de două cifre, al minutului
- `SS` este identificatorul numeric, de două cifre, al secundei
- `TIP_EVENIMENT` este identificatorul tipului evenimentului
- `MESAJ` este mesajul specific evenimentului.

```
[..]
[29.11.2020 23:59:59] INFO: Aranea was launched.
[..]
```

Exemplu 3: Extras dintr-un fișier de jurnalizare

## 2.2 Structuri de Date

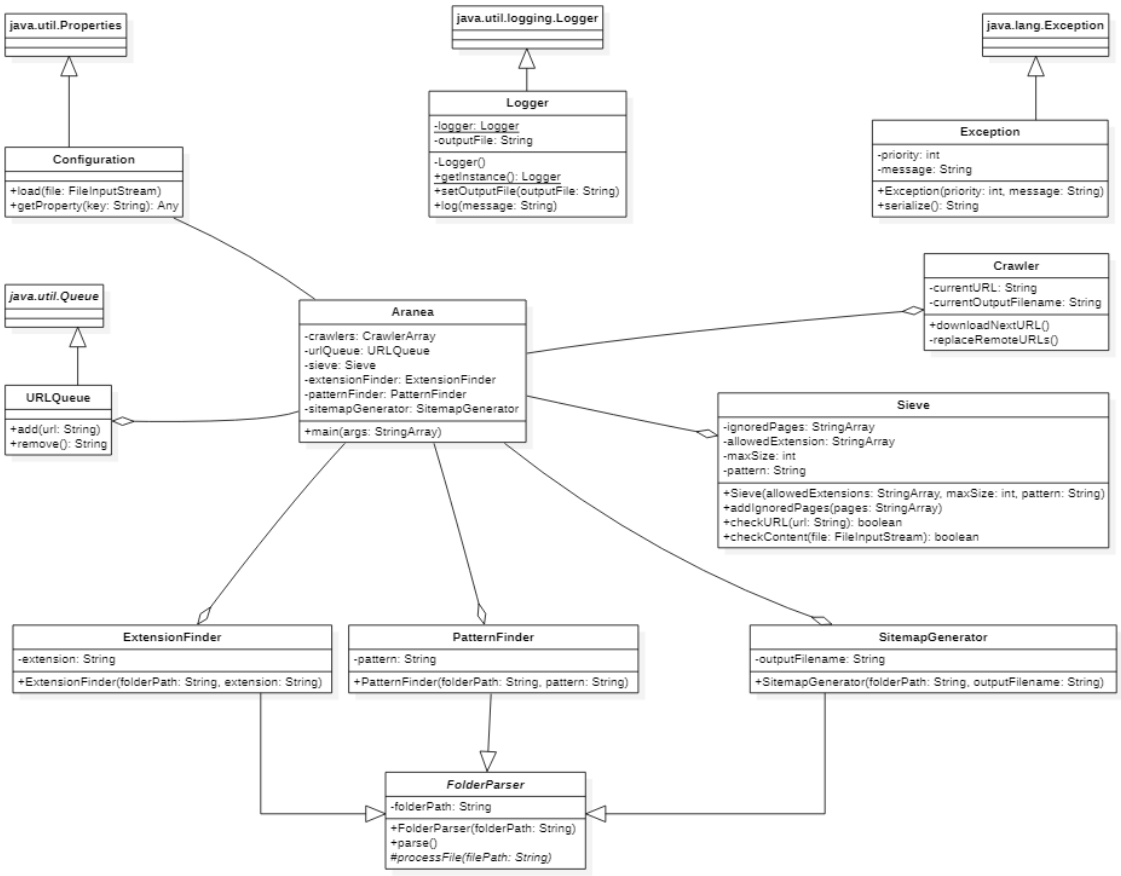
Numai două structuri de date apar în cadrul programului. Ele sunt membrii ale clasei principale, `Aranea`, și implementate cu ajutorul unor clase specifice Java.

| Nume | Clasă de Implementare | Descriere | Acțiuni Posibile |
|---|---|---|---|
| `URLQueue` | `java.util.Queue` | Fiind o coadă, salvează referințe către pagini ce trebuiesc verificate și descărcate. | 1. Adăugarea unei pagini folosind metoda `add`<br>2. Extragerea unei pagini folosind metoda `remove` |
| `Configuration` | `java.util.Properties` | Salvează configurația primită prin intermediul fișierului de intrare, de tip `CONFIG_FILE`. | 1. Inițializarea pe baza unui fișier, folosind metoda `load`<br>2. Citirea valorii salvate la o anumită cheie, folosind metoda `getProperty` |

Tabel 2: Structuri de date globale

# 3 Proiectare a Arhitecturii

## 3.1 Diagramă de Clase




Imagine 1: Diagramă de clase

În diagramă, tipurile de date `DataArray` sunt o notație pentru `Data[]`. În plus, legăturile de asociere dintre anumite clase (`Configuration`, `Logger` și `Exception`), cât și clasele ce moștenesc `Exception`, au fost omise pentru a nu încărca diagrama.

## 3.2 Descriere a Claselor

În tabelele următore, clasele `URLQueue` și `Configuration` au fost omise întrucât acestea au fost detaliate în cel cu structuri de date globale.

| | |
|---|---|
| Nume | `Logger` |
| Detalii despre Implementare | clasă Singleton, moștenire a clasei `java.util.loggin.Logger` |
| Descriere | Jurnalizează într-un fișier evenimentele întâmplate de-a lungul execuției. Poate fi folosită de către orice altă componentă pentru a avea acces la funcționalitatea de jurnalizare. |
| Acțiuni Posibile | 1. Preluarea instanței globale prin metoda `getInstance`<br>2. Setarea unui fișier de jurnalizare prin metoda `setOutputFile`<br>3. Jurnalizarea unui eveniment prin metoda `log` |

Tabel 3: Descriere a clasei `Logger`

| | |
|---|---|
| Nume | `Exception` |
| Detalii despre Implementare | moștenire a clasei `java.lang.Exception` |
| Descriere | Reprezintă o excepție, putând fi moștenită în diferite alte clase ce descriu tipuri particulare de excepții. |
| Acțiuni Posibile | 1. Instanțierea și setarea priorității și a mesajului, prin intermediul constructorului<br>2. Serializarea informațiilor prin metoda `serialize`, ce presupune crearea unui șir de caractere cu formatul *`"TIP_EVENIMENT: MESAJ"`*, unde:<br>• `TIP_EVENIMENT` este *`"INFO"`* dacă prioritatea setată este 0, *`"WARNING"`* dacă prioritatea este 1, *`"ERROR"`* dacă prioritatea este 2 și *`"CRITICAL"`* dacă prioritatea este 3<br>• `MESAJ` este mesajul specific excepției. |

Tabel 4: Descriere a clasei `Exception`

| | |
|---|---|
| Nume | `FolderParser` |
| Detalii despre Implementare | clasă abstractă |
| Descriere | Parcurge recursiv un director și efectuează o anumită operațiune pe baza fiecărui fișier găsit. |
| Acțiuni Posibile | 1. Instanțierea și setarea directorului prin intermediul constructorului<br>2. Începerea procesării prin metoda `parse`<br>3. Procesarea apelată la găsirea unui fișier, prin metoda virtuală `processFile` |

Tabel 5: Descriere a clasei `FolderParser`

| | |
|---|---|
| Nume | `ExtensionFinder` |
| Detalii despre Implementare | implementare (prin moștenire) a clasei abstracte `FolderParser` |
| Descriere | Parcurge recursiv un director și printează pe ecran numele fișierelor având o anumită extensie. |
| Acțiuni Posibile | 1. Instanțierea și setarea directorului și a extensiei prin intermediul constructorului |

Tabel 6: Descriere a clasei `ExtensionFinder`

| | |
|---|---|
| Nume | `PatternFinder` |
| Detalii despre Implementare | implementare (prin moștenire) a clasei abstracte `FolderParser` |
| Descriere | Parcurge recursiv un director și printează pe ecran numele fișierelor având conținut ce potrivește un anumit șablon. |
| Acțiuni Posibile | 1. Instanțierea și setarea directorului și a șablonului prin intermediul constructorului |

Tabel 7: Descriere a clasei `PatternFinder`

| | |
|---|---|
| Nume | `SitemapGenerator` |
| Detalii despre Implementare | implementare (prin moștenire) a clasei abstracte `FolderParser` |
| Descriere | Parcurge recursiv un director și creează o hartă, pe care o scrie, la finalizarea operației, într-un fișier. |
| Acțiuni Posibile | 1. Instanțierea și setarea directorului și a fișierului de ieșire prin intermediul constructorului |

Tabel 8: Descriere a clasei `SitemapGenerator`

| | |
|---|---|
| Nume | `Sieve` |
| Detalii despre Implementare | ∅ |
| Descriere | Dă permisiunea ca un fișier să fie descărcat sau salvat local, pe baza unor criterii. Pe baza URL-ului, verifică dacă fișierul se află în `robots.txt`, extensia și dimensiunea (printr-o cerere de tip `HEAD`). În plus, poate verifica dacă un anumit șablon este potrivit de conținutul fișierului. |
| Acțiuni Posibile | 1. Instanțierea și setarea extensiilor permise, a dimensiunii maxime și a unui șablon prin intermediul constructorului<br>2. Adăugarea unor fișiere ce se află în `robots.txt`, prin metoda `addIgnoredPages`<br>3. Verificarea unui URL prin metoda `checkURL`<br>4. Verificarea conținutului unui fișier prin metoda `checkContent` |

Tabel 9: Descriere a clasei `Sieve`

| | |
|---|---|
| Nume | `Crawler` |
| Detalii despre Implementare | clasă tip worker (fiecare instanță rulează pe un fir de execuție separat) |
| Descriere | Descarcă pagini salvate în coada de tip `URLQueue`. După salvarea lor, parcurge conținutul fișierului și înlocuiește toate URL-urile externe întâlnite cu unele către fișierel locale. De specificat este faptul că toate cererile vor fi efectuate cu un agent specific, `AraneaBot`. |
| Acțiuni Posibile | 1. Descărcarea unei pagini prin metoda `downloadNextURL`<br>2. Înlocuirea URL-urilor din fișierele savate prin metoda privată `replaceRemoteURLs` |

Tabel 10: Descriere a clasei `Crawler`

| | |
|---|---|
| Nume | `Aranea` |
| Detalii despre Implementare | ∅ |
| Descriere | Este inițializată la apelarea metodei `main`, pe baza parametrilor programului. Cuprinde majoritatea celorlaltor clase descrise, orchestrându-le astfel încât execuția să fie de succes:<br>• pentru operațiunea de crawling<br>– inițializează configurația de tip `Configuration`, pe baza fișierului de configurare<br>– inițializează coada de pagini, de tip `URLQueue`, pe baza fișierului de intrare și a fișierelor `robots.txt` specifice lor<br>– inițializează un filtru pentru paginile descărcate, prin intermediul clasei `Sieve`<br>– creează fire de execuție pentru fiecare instanță a clasei `Crawler`<br>– dacă se dorește crearea de hărți, instanțiază un obiect de tipul `SitemapGenerator`<br>• pentru operațiunea de căutare a fișierelor cu o anumită extensie, instanțiază un obiect de tipul `ExtensionFinder`<br>• pentru operațiunea de căutare a fișierelor cu conținut ce potrivește un anumit șablon, instanțiază un obiect de tipul `PatternFinder` |
| Acțiuni Posibile | ∅ |

Tabel 11: Descriere a clasei `Aranea`

# 4 Modelare a Interfeței cu Utilizatorul

Interfațarea cu utilizatorul se realizează prin intermediul liniei de comandă. Cum programul este distribuit sub forma unei arhive JAR, acesta poate fi rulat cu ajutorul comenzii `java -jar ABSOLUTE_PATH_TO/aranea.jar COMMAND [ARGUMENTS]`. Din cauza dificultății ei, o adăugare a unui alias va fi recomandată utilizatorilor, cu ajutorul uneia dintre comenzile următoare:

- `doskey aranea=java -jar ABSOLUTE_PATH_TO/aranea.jar` pentru sistemele de operare Windows
- `alias aranea=java -jar ABSOLUTE_PATH_TO/aranea.jar` pentru sistemele de operare Linux și macOS.

Utilizatorii vor putea rula următoarele comenzi pentru a folosi Aranea:

- `aranea crawl URLS_FILE CONFIG_FILE` pentru executarea crawling-ului pe o serie de website-uri
- `aranea crawl list EXTENSION` pentru listarea fișierelor salvate local, cu o anumită extensie
- `aranea crawl search PATTERN` pentru căutarea unui șablon în paginile salvate local
- `aranea` sau `aranea help` pentru soliciarea ajutorului.

În funcție de operațiune, utilizatorilor li se poate afișa text pe ecran sau oferi fișiere locale pe care să le verifice manual, ulterior.

# 5 Elemente de Testare

Testarea aplicației rezultate în urma perioadei de implementare va fi realizată conform cu următoarea metodologie:

1. Crearea a două teste automate (unul cu dificultate scăzută și unul cu dificultate ridicată) pentru fiecare caz de utilizare, cu ajutorul JUnit
2. Rularea testelor și observarea rezultatelor
3. Consemnarea funcționării incorecte a aplicației în secțiunea de probleme a Github și rezolvarea individuală sau colaborativă a problemei.