Tema 2 - File System

- Responsabili: Laurentiu Piciu Theodor Stoican
- **Deadline** : 29.11.2016 (ora 23:55)
- Data publicării: 14.11.2016
- Data ultimei modificări: 23.11.2016
  - Am adăugat la resurse, pentru claritate, un exemplu efectiv (Test demo)
  - Am adaugat checker-ul, cu teste locale.
  - Am modificat erorile apărute în unele fișiere de intrare / de referință
- Data adăugării tester-ului: 17.11.2016

Obiective

- Aprofundarea noțiunilor de moștenire, agregare și interfațare în contextul programării orientate pe obiecte.
- Utilizarea unor design patterns în contextul implementării unei aplicații reale.
- Înțelegerea modului de funcționare a unui sistem de fișiere.
- Respectarea unui coding-style adecvat.

Cerințe

Va trebui să implementați propriul vostru sistem de fișiere minimal în stilul celor folosite pe platformele de tip **Unix**. Odată creat sistemul de fișiere, acesta va trebui să suporte o parte din comenzile cele mai des utilizate în sisteme de operare tip Unix (ex: ls, cd, mkdir, touch etc.).

Pentru aceasta, va trebui să folosiți următoarele design patterns:

- **Composite Pattern**, care va fi folosit pentru crearea structurii arborescente a sistemului de fișiere.
- **Factory Pattern** (despre care găsiți noțiuni și în **laboratorul 11**), care va fi folosit pentru gestionarea creării fiecărei comenzi în parte.

Dacă tema nu respectă cele două restricții de design, nu vă va fi acordat punctajul aferent testelor.

Descriere

Composite Pattern

Sistemul de fișiere va fi implementat sub forma unui arbore. Pentru a realiza acest lucru, se va folosi Composite Pattern, un pattern structural, care implică crearea unei clase în care se agregă instanțe ale aceleiași clase. Această agregare, așa cum am specificat mai sus, va genera un arbore (ce va conține, la un moment dat, și noduri frunză (le vom numi primitive), indiferent de dimensiunea sa). La nivelul implementării, vom dori sa tratăm uniform cazurile când avem de-a face cu primitive sau noduri intermediare (de obicei, cei care implementează se ocupă separat de cele două tipuri de noduri). În acest sens, Composite Pattern se referă la existența unui obiect, creat ca o compunere din mai multe obiecte, ce prezintă functionalitați similare, de aici provenind, de fapt, și funcționalitatea cheie a acestui pattern (manipularea unei singure instanțe se realizează la fel ca și manipularea unui grup de instanțe).În cazul de față, pentru a înțelege și mai bine detaliile teoretice, se poate crea o clasă cu rol de părinte (o numim generic Entitate) atât pentru directoare, cât și pentru fișiere. Clasa Director va agrega alte instanțe de tip Entitate.

Structura arborescentă, transpusă vizual, ar trebui să arate astfel:

```
/
   dir1
      file11
      dir12
         file121
         file122
   dir2
      dir21
   dir3
      file31
```

Se poate observa faptul că fiecare director poate avea ca descendenți și alte entități, însă fișierele vor fi întotdeauna frunze.

Factory Pattern

Pentru a genera instanțe ale comenzilor de care avem nevoie, vom folosi **Factory Pattern**. Ca să ilustrăm într-un mod cât mai concret acest lucru, ne putem imagina că avem o “fabrică” (un obiect de tip Factory) care primește ca parametru tipul comenzii dorite și returneaza o instanță a acelei comenzi.

Implementarea comenzilor

Fiecare comandă va fi citită dintr-un fișier de intrare. Într-un fișier de ieșire vor fi afișate eventuale mesaje de eroare, iar după ce se citesc toate comenzile, în același fișier de ieșire va fi afișat întreg arborele creat. Erorile pot apărea în urma unor acțiuni nepermise ce se pot executa și fiecare eroare, împreună cu mesajul asociat, vor fi descrise în cele ce urmează:

```
-1: <command>: Is a directory
-2: <command>: No such directory
-3: <command>: Not a directory
```

```
-4: <command>: No rights to read
-5: <command>: No rights to write
-6: <command>: No rights to execute
-7: <command>: File already exists
-8: <command>: User does not exist
-9: <command>: User already exists
-10: <command>: No rights to change user status
-11: <command>: No such file
-12: <command>: No such file or directory
-13: <command>: Cannot delete parent or current directory
-14: <command>: Non empty directory
```

Permisiuni

După cum bine știți, sistemele de fișiere înglobează o logică de **permisiuni** bine pusă la punct. În sistemul nostru, vom ține cont doar de două tipuri de permisiuni (vom exclude noțiunea de **grup** din aplicație):

- Permisiuni ale utilizatorului ce **deține** fișierul/directorul
- Permisiuni ale altor utilizatori din sistem

Permisiunile implicite ale entităților noi create vor fi: **rwx** (pentru utilizatorul ce creează și, evident, deține entitatea) și **- - -** (pentru ceilalți utilizatori).

Pentru `/` , permisiunile implicite vor fi **rwxr-x** , pentru ca ceilalți utilizatori să poată trece prin `/` și să poată lista conținutul acestuia. Se garantează că permisiunile pentru `/` vor rămâne aceleași pe tot parcursul rulării programului.

Comanda adduser

Inițial, în sistemul nostru există un singur utilizator (**root**), iar directorul asignat lui va fi `/` . Acesta va deține drepturi absolute peste orice director/fișier din sistem.
Comanda va adăuga un nou utilizator în sistem și în plus va crea automat un director asignat utilizatorului care va avea numele utilizatorului și care va fi descendent al directorului `/` .
Exemplu:

```
adduser andrei
adduser alexandru
```

Arborele va arăta în felul următor:

```
/
   andrei
   alexandru
```

Cazurile speciale care pot fi întâlnite:

```
-10: <command>: No rights to change user status - dacă utilizatorul care
```

```
adaugă este altul decât root
-9: <command>: User already exists - dacă se încearcă adăugarea unui
utilizator care deja există în sistem
```

Comanda deluser

Utilizatorii pot fi șterși la un moment dat din sistem (operație permisă doar **root**-ului). În cazul în care se întâmplă acest lucru, fiecare entitate deținută de utilizatorul șters va trebui sa își seteze pentru acest câmp un alt deținător (în cazul nostru, îl vom alege pe cel care a fost adăugat primul cu ajutorul comenzii **adduser**).

Cazuri speciale:

```
-10: <command>: No rights to change user status - dacă utilizatorul care
adaugă este altul decât root
-8: <command>: User does not exist - dacă utilizatorul care se dorește a fi
șters nu există în sistem
```

Comanda chuser

Cu ajutorul acesteia vom putea să modificăm utilizatorul activ în sistem. În acest caz, directorul curent se va modifica în directorul care i s-a asignat la creare.

Noul utilizator poate fi chiar și **root**-ul. Nu există nicio restricție în acest sens.

Cazuri speciale:

```
-8: <command>: User does not exist - dacă utilizatorul care se dorește a fi
șters nu există în sistem
```

Comanda cd

Comanda primește ca parametru o cale (**relativă** sau **absolută**). În funcție de calea primită, ulilizatorul își va modifica directorul curent.
Exemplu:

```
cd /andrei/tema1/.././poo - calea absolută începe obligatoriu de la /
cd andrei/./../abcd - calea este relativă la directorul curent
```

Cazuri speciale:

```
-3: <command>: Not a directory - dacă în calea pe care o precizăm apare la un
moment dat un fișier
-2: <command>: No such directory - dacă în calea pe care o precizăm apare la
un moment dat un nume de entitate care nu există
```

```
-6: <command>: No rights to execute - dacă nu avem drepturi de a executa pe un director în care dorim să intrăm
```

Comenzile pe care urmează să le descriem în continuare vor avea la rândul lor, ca parametru, o anumită cale (**asemenea ca la comanda cd**). În implementarea voastră trebuie să țineți cont că pentru a ajunge la o anumită cale, pot apărea din nou cazurile speciale descrise pentru **comanda cd**.

Comanda mkdir

Cu ajutorul acestei comenzi vom crea noi directoare. Ea va primi ca parametru calea noului director.

Cazuri speciale:

```
mkdir <path>

-1: <command>: Is a directory - dacă noul director ce se dorește a fi creat deja există
-3: <command>: Not a directory - dacă deja există la calea specificată un fișier cu numele pe care dorim să îl dăm directorului
-5: <command>: No rights to write - dacă utilizatorul nu are drept de scriere asupra directorului în care vrem sa creăm noua entitate
```

Comanda ls

Comanda listează conținutul unui directorului specificat în calea dată ca parametru. Mai mult, comanda poate primi ca parametru și un fișier, iar în acest caz se va respecta șablonul dat mai jos. Rezultatul ce va trebui scris în fișierul de ieșire, în cazul în care comanda se execută cu succes, trebuie să arate astfel:

- pentru directoare:

```
<directory_name> dr-xrw- <owner>
```

- pentru fișiere:

```
<file_name> frwx--- <owner>
```

Permisiunile afișate vor trebui să fie cele reale pentru o anumită entitate. Dacă nu există setată o anumită permisiune, se va afișa caracterul **-**

În cazul în care comanda eșuează, se vor afișa mesajele de eroare, după cum urmează:

```
ls <path>

-12: <command>: No such file or directory - dacă nu va fi găsită nicio enitate la calea specificată
```

```
-4: <command>: No rights to read - dacă utilizatorul activ nu are drept de citire asupra entității
```

Comanda chmod

Comanda va primi 2 parametri:

- un număr de două cifre (fiecare cifră va avea valoarea maxim 7) care reprezintă permisiunile pentru cel ce deține entitatea, respectiv pentru ceilalți utilizatori. Pentru a vă reaminti codificarea fiecărei cifre, puteți citi mai multe despre comanda **chmod**
- o cale către o entitate pentru care dorim să realizăm modificările

Exemplu:

```
chmod 54 myfile.txt
```

va seta permisiunile **r-xr- -** pentru fișierul **myfile.txt** din directorul curent.

Cazuri speciale:

```
-12: <command>: No such file or directory - dacă nu va fi găsită nicio enitate la calea specificată
-5: <command>: No rights to write - dacă utilizatorul curent nu are drept de scriere asupra entității
```

Comanda touch

Cu ajutorul acestei comenzi vom crea noi fișiere. Ea va primi ca parametru calea noului fișier.

Cazuri speciale:

```
touch <path>

-1: <command>: Is a directory - dacă un director cu același nume există la calea specificată
-7: <command>: File already exists - dacă un fișier cu același nume există la calea specificată
-5: <command>: No rights to write - dacă utilizatorul activ nu are drept de scriere asupra directorului în care vrem să creăm
```

Comanda rm

Această comandă poate exista în două forme:

- **rm <path>** - șterge fișierul specificat la calea dată ca argument
- **rm -r <path>** - șterge întreaga ierarhie de fișiere începând cu **entitatea** specificată la calea dată

ca argument

Cazuri speciale:

```
rm <path>
-1: <command>: Is a directory - dacă un director cu același nume există la calea specificată
-11: <command>: No such file - dacă la calea specificată nu există fișierul pe care dorim sa îl ștergem
-5: <command>: No rights to write - dacă utilizatorul activ nu are drept de scriere asupra directorului din care vrem sa ștergem

rm -r <path>
-12: <command>: No such file or directory - dacă la calea specificată nu există entitatea pe care dorim să o ștergem
-13: <command>: Cannot delete parent or current directory - dacă se încearcă ștergerea părintelui, sau a vreunui strămoș (în ierarhia de fișiere), sau a directorului curent
-5: <command>: dacă utilizatorul activ nu are drept de scriere asupra directorului din care vrem sa ștergem
```

Comanda rmdir

Comanda primește ca parametru o cale și șterge directorul specificat **doar dacă** acesta nu are descendenți (e gol).

Cazuri speciale:

```
rmdir <path>

-3: <command>: Not a directory - dacă deja există la calea specificată un fișier cu numele pe care dorim să îl dăm directorului
-13: <command>: Cannot delete parent or current directory - dacă se încearcă ștergerea părintelui, sau a vreunui strămoș (în ierarhia de fișiere), sau a directorului curent
-14: <command>: Non empty directory - dacă directorul pe care dorim să îl ștergem nu este gol
-5: <command>: No rights to write - dacă utilizatorul activ nu are drept de scriere asupra directorului din care vrem sa ștergem
-2: <command>: No such directory - dacă directorul pe care dorim sa îl ștergem nu se află la calea specificată
```

Comanda writetofile

Fiecare fișier va avea un conținut (text). Această comandă e menită să seteze un anumit conținut pentru fișierul care este specificat prin calea dată ca parametru.

Forma comenzii și cazurile speciale:

```
writetofile <path> <content>

-1: <command>: Is a directory - dacă un director cu același nume există la calea specificată
-11: <command>: No such file - dacă la calea specificată nu există fișierul pe care dorim sa îl ștergem
-5: <command>: No rights to write - dacă utilizatorul activ nu are drept de scriere asupra fișierului
```

Comanda cat

Afișează în fișierul de ieșire conținutul fișierului care este specificat prin calea dată ca parametru.

Forma comenzii și cazurile speciale:

```
cat <path>

-1: <command>: Is a directory - dacă un director cu același nume există la calea specificată
-11: <command>: No such file - dacă la calea specificată nu există fișierul pe care dorim să îl citim
-4: <command>: No rights to read - dacă utilizatorul activ nu are drept de citire asupra fișierului
```

Date de intrare. Date de ieșire.

Intrare

Programul vostru va deține o clasă în care va exista o metodă main, din care se va dirija fluxul derulării comenzilor pe arborele vostru. Va trebui să parsați comenzile din fișierul de intrare și să le executați, una câte una, până la finalul fișierului. Un fișier de intrare, va conține, pe fiecare linie, care respectă următorul format:

```
<nume_comanda> <parametru> <argument> (ex: mkdir "/johndoe/home")
```

Fiecare fișier din suita de teste va conține, ca primă linie, **mkdir "/" (o convenție folosită pentru a păstra formatul de tip Unix a sistemului de fișiere).**

Recomandarea noastră ar fi să NU folosiți **Scanner** pentru citirea din fișiere. La această **adresă** găsiți exemple utile care vă arată ce alte alternative aveți.

Ieșire

Va trebui sa afișati rezultatul comenzilor voastre (în cazul în care acestea generează erori, sub forma codului de eroare corespunzător, altfel nu ar trebui sa afișați nimic), delimitat de următorul rezultat prin caracterul **'\n'**. De asemenea, după afișarea rezultatelor comenzilor, va trebui printat și sistemul de fișiere efectiv, sub forma unui arbore, ce respecta următorul format:

```
/ drwxr-x root
    etc drwxrw- john
    bin dr---w- john
    home dr-x-wx andrew
          mystuff drwx--- andrew
                    faculty.pdf f--xrwx andrew
                    personal.xml fr--r-- john
          hallelujah drwx-w- andrew
    usr d-w-rwx john
    var d---rw- john
    dev d-wx-wx john
```

Pentru tot ceea ce va trebui să afișați, veți folosi comnda **System.out.println**, care afișează la **stdout**.

Pentru a evidenția structura arborescentă, vom indenta intrările cu câte un tab ('\t'), relativ față de părintele nodului curent.

Radacina arborelui va porni de la nivelul de identare 0.
De asemenea, observăm că o intrare în arbore are următorul format:

```
<file_name> <type_of_entry_and_permissions> <owner>
```

Cele 3 câmpuri sunt separate prin spații, iar câmpul al doilea din această intrare e compus din 7 caractere cu următoarea semnificație:

- prima literă reprezintă tipul intrării (poate fi “d” - pentru directoare și “f” - pentru fișiere)
- următoarele 3 litere reprezintă permisiunile pentru owner-ul directorului/fișierului respectiv
- ultimele 3 litere definesc permisiunile de tip “Others”

Primul câmp reprezintă numele fișierului/directorului, iar ultimul câmp conține numele deținătorului (owner-ul) fișierului/directorului respectiv.

Punctaj

- **90p** teste publice
- **10p** README, comentarii, JavaDocs, coding style

Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va duce la **anularea punctajului** de pe parcursul semestrului și **repetarea materiei** atât pentru sursă(e) cât și pentru destinație(ii), fără excepție.

Structura arhivei

Arhiva pe care o veţi urca pe **Vmchecker** va trebui să conţină în directorul rădăcină:

- `README` în care să explicaţi
  - cum aţi gândit implementarea comenzilor
  - cum aţi creat design-ul de intrări într-un director (fișiere și directoare), incluzând și Composite Pattern
- alte detalii relevante pentru implementare
- directorul `src` cu fişiere sursă
- directorul `doc`, generat de javadoc

Comentariile tuturor claselor și metodelor relevante vor trebui să respecte formatul Javadoc.

Nu uitați de paginile wiki: indicatii pentru teme și coding style.

Resurse

- PDF enunţ temă
- Test demo
- **Checker + teste publice**: Checker v1.2

Referințe

- File System
- File System - OS perspective
- Composite Pattern tutorial
- Composite Pattern
- Reading a plain text file in Java
- StringTokenizer
- Laboratorul 4 - Factory
- Indicatii pentru teme
- Recomandari coding style