Tema 2 - VCS (Version Control System)

- Responsabili: Mihai Burduselu, Vlad Alexandrescu,Tiberiu Lepadatu
- Deadline: ~~21.11.2018~~ **25.11.2018**
- Deadline hard: ~~28.11.2018~~ **04.12.2018**
- Data publicării: 07.11.2018
- Data ultimei modificări: 07.11.2018

Obiective

- Aprofundarea noțiunilor de moștenire, agregare și interfațare în contextul programării orientate pe obiecte.
- Utilizarea unor design patterns în contextul implementării unei aplicații reale, și anume:
  - Singleton Pattern
  - Factory Pattern
- Respectarea unui **coding-style** adecvat.

Cerințe

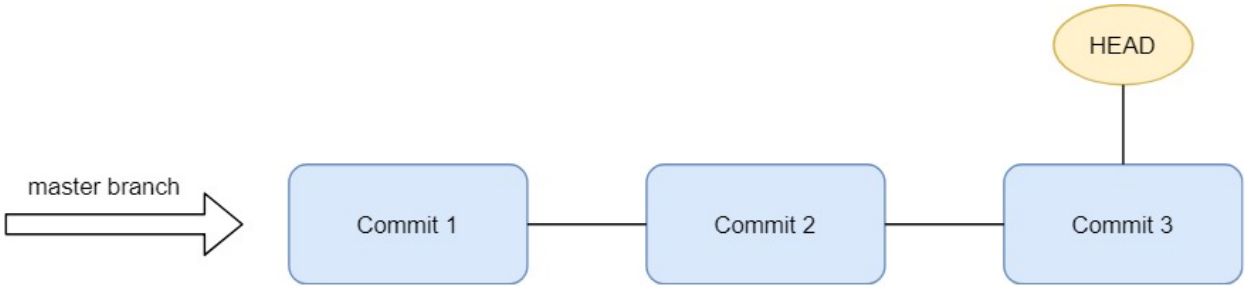
Se dorește implementarea unui sistem de versionare. Aplicația trebuie să suporte un set minim de comenzi **unix** pentru a putea crea, modifica si șterge fișiere, cât și un set de comenzi **vcs** care va permite salvarea stării curente a sistemului de fișiere. Scopul final al acestui program este de a ne putea întoarce la versiuni anterioare ale sistemului de fișiere.
Întrucât tema se axează pe implementarea unui sistem de versionare, partea de comenzi **unix** este inclusă în schelet.

Descriere

VCS (**Version Control System**) permit gestionarea versiunilor multiple ale fișierelor. Fiecare modificare a unui fișier este înregistrată în sistem. O astfel de înregistrare poartă numele de **commit**. In cadrul temei cand vom spune că am dat un **commit**, înseamnă că am salvat versiunea curenta a tuturor fișierelor din sistemul de fișiere.
La finalul implementării aplicația trebuie sa ne permită sa dăm commituri și să putem să revenim la o versiune anterioară a sistemului de fișiere, adică să ne întoarcem câteva commit-uri în trecut.

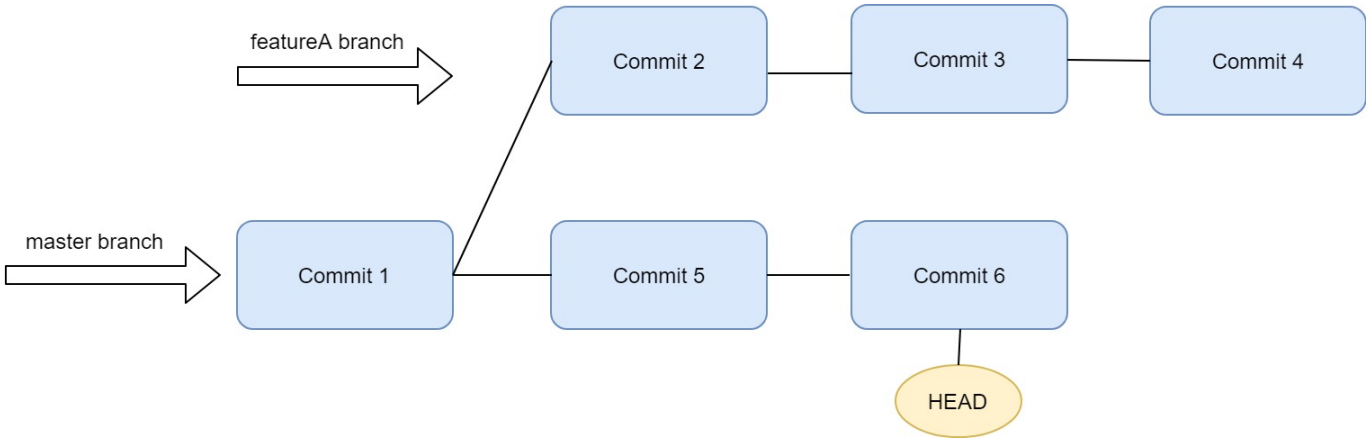
Cursorul HEAD

La capătul acestui flow de commit-uri se află un cursor denumit **HEAD**. În momentul adăugării unui nou commit sau întoarcerii la un commit anterior, cursorul **HEAD** se mută pe commit-ul respectiv.



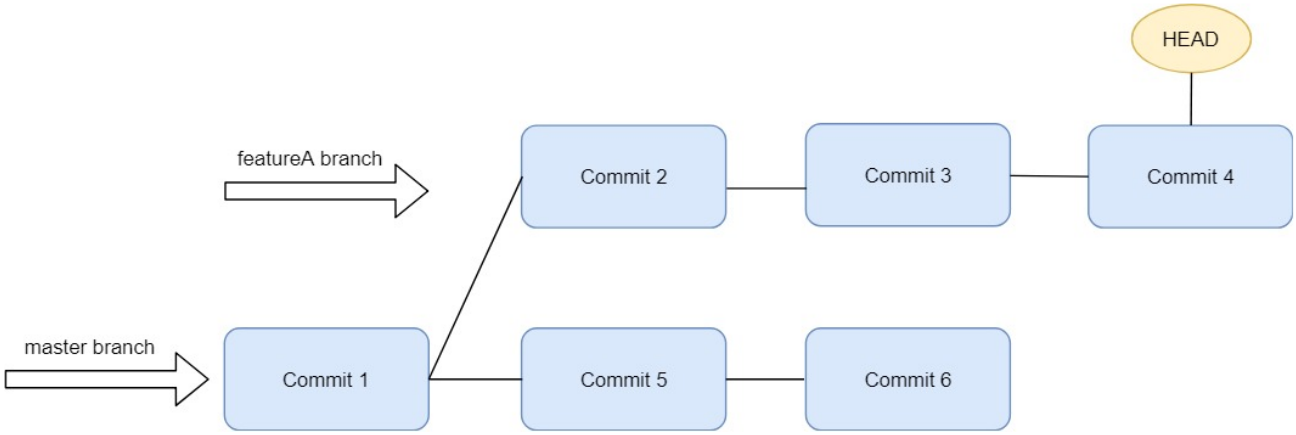
Semnificația unui branch

Sistemele de versionare permit si lucrul mai multor persoane pe un set de fișiere. Astfel, a apărut noțiunea de **branch**. **Branch-ul** este o ramură a sistemului de fișiere. Practic, putem sa avem același sistem de fișiere duplicat pe mai multe branch-uri ceea ce ne permite ca fiecare dezvoltator să lucreze pe branch-ul său.



Branch-ul master

Branch-ul principal se numește **master**. În momentul în care ne mutăm de pe un branch pe altul efectuăm o operație de checkout. Astfel, pointerul **HEAD** este mutat pe branch-ul pe care dorim să ajungem.



Staging

Cand întâlnim notiunea de “Staged changes”, ne referim la operațiile care încă nu au fost “commit-uite” si care
au alterat starea sistemului de fișiere.
De exemplu, daca de la ultima comandă de commit am mai creat un fișier, această operație de touch va fi adăugată în **staging**.
În momentul în care se execută operația **vcs commit**, atunci staging-ul va fi golit, iar HEAD-ul se va muta pe noul commit.
În momentul în care se execută operația **vcs rollback**, atunci staging-ul va fi golit, HEAD-ul va rămâne pe commit-ul curent, iar activeFileSystemSnapshot va reveni la valoarea din commit-ului curent.

Referință **FOARTE UTILĂ:** https://imgflip.com/i/2lveks

Implementare

Tema este împărțită în 2 părți: operațiile de filesystem și cele de vcs.
Fiecare comandă va fi citită dintr-un fișier de intrare și va genera un output sau un mesaj de eroare ce va fi scris în fisierul de iesire.
Entry point-ul temei va fi clasa Main din schelet. Metoda main va primi ca parametru de la checker fișierele de intrare si ieșire.

Rularea aplicației începe prin inițializarea vcs-ului și crearea unui prim branch numit **“master”** și a unui prim commit cu id-ul 3 si mesajul **“First commit”** (id-ul 3 pe primul commit este assignat automat daca folositi IDGenerator din schelet).
Abia după rularea internă a acestor operații programul va incepe să citească linie cu linie inputul și să interpreteze operațiile.

Fișierele si directoarele create în cadrul temei vor fi **virtuale**. Ele vor fi doar niste obiecte reținute în memorie fără vreo reprezentare pe disc.

Mesaje de eroare

În cadrul temei sunt 5 tipuri de mesaje de eroare:

- -1: Vcs: Bad command
- -2: Vcs: Bad path
- -3: Vcs: There are staged operations, please do commit/rollback!
- 1: System: Bad command
- 2: System: Bad path

Pentru comenzile date cu sintaxa greșită se va afișa mesajele -1: Vcs: Bad command sau 1: System: Bad command.

Exemplu de comenzi cu sintaxă greșită: vcs st, vcs commit (fara -m mesaj), touch fara parametru, cd fara parametru, etc.

Unele comenzi de mai jos sunt întâlnite în sistemul de versionare git, însă pentru a ușura implementarea temei am modificat funcționalitatea lor. Citiți cu atenție descrierea fiecărei comenzi în parte.

Comenzi filesystem

Comanda cd

Comanda primește ca parametru o cale (relativă sau absolută). În funcție de calea primită, se va va modifica directorul curent.

Exemplu:

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

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un fișier;
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există

Comanda mkdir

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

Exemplu:

```
mkdir A
mkdir ../B/C
```

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apar erorile menționate la comanda cd
  - dacă noul director ce se dorește a fi creat deja există)))
- **1: System: Bad command**
  - dacă deja există la calea specificată un fișier cu numele pe care dorim să îl dăm directorului

Comanda ls

Comanda listează conținutul unui directorului specificat în calea dată ca parametru.

Exemplu:

```
ls /root
ls
```

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există
- **1: System: Bad command**
  - dacă se dă comanda ls pe un fișier

Comanda touch

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

Exemplu:

```
touch path/a
```

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există

Comanda rm

Această comandă poate exista în 3 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
- **rmdir <path>** - sterge directorul specificat la calea dată ca argument (directorul nu trebuie neapărat să fie gol).

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există
- **1: System: Bad command**
  - dacă se dă comanda rm pe un director
  - dacă se dă comanda rmdir pe un fișier

Comanda writetofile

Fiecare fișier va avea un conținut (text). Această comandă e menită să lipească un anumit conținut la finalul fișierului care este specificat prin calea dată ca parametru.

Exemplu:

```
writetofile path/file1 acesta este continutul fisierului
```

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există
  - dacă la calea specificată nu există fișierul pe care dorim sa îl modificăm
  - dacă la calea specificată se află un director

Comanda cat

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

Exemplu:

```
cat file1
cat /root/poo/Main.java
```

Mesaje de eroare:

- **2: System: Bad path**
  - dacă în calea pe care o precizăm apare la un moment dat un nume de entitate care nu există
  - dacă în calea pe care o precizăm nu există fișierul pe care dorim să îl citim
  - dacă un director cu același nume există la calea specificată

Comanda print

Comandă auxiliară pentru afișarea întregii ierarhii de fișiere.

Exemplu:

```
print
```

Comenzi VCS

Comanda status

Comanda listează numele branch-ului pe care ne aflăm și modificările care sunt in staging.
“Staged changes” reprezintă toate modificările și operațiile aplicate pe filesystem de la ultimul commit pană in prezent. Staging-ul se golește în momentul în care se dau comanzile commit sau rollback.
Operatiile track-uite de vcs status: mkdir, touch, writetofile, rm, rmdir, cd.

Exemplu:

```
vcs status
```

Exemplu output:

```
On branch: master
Staged changes:
   Created directory A
   Created file ana
   Added "anaaremere" to file ana
   Removed A
   Removed ana
   Changed directory to dir1
```

Comanda branch

Crează un nou branch cu aceeași structură a sistemului de fișiere ca cea descrisă de commit-ul branch-ului curent.

Exemplu:

```
vcs branch tema2V1
```

Mesaje de eroare:

- **-1: Vcs: Bad command**
  - dacă există deja un branch cu același nume

Comanda commit

Genereaza un nou commit, prin aplicarea modificarilor puse in staging commit-ului curent. Noul commit va avea commit-ul curent ca parinte.

Exemplu:

```
vcs commit -m Add list items functionality
```

Mesaje de eroare:

- **-1: Vcs: Bad command**
  - dacă nu există nicio operație in staging

Comanda checkout

Are 2 functionalități:

1. poate muta pointerul HEAD pe un al branch
2. poate muta pointerul HEAD pe un commit anterior de pe branch-ul curent

Dupa ce ne mutam pe un commit anterior (sa il denumim commitul x), toate commiturile care au fost date dupa commitul x se vor sterge. Id-urile deja generate nu vor mai fi folosite.

De exemplu, daca avem commit-urile cu id-urile 3, 5, 7, 9 (noi aflandu-ne pe ultimul commit), prin operatia de vcs checkout -c 3, se vor sterge commit-urile 5, 7, 9. In momentul in care dam un nou commit, acesta va avea id-ul 11.

Forma comenzii:

```
vcs checkout branchName
vcs checkout -c commitId
```

Exemplu:

```
vcs checkout tema2V1
vcs checkout -c 3
```

Mesaje de eroare:

- **-1: Vcs: Bad command**
  - dacă nu există un branch cu numele specificat
- **-2: Vcs: Bad path**
  - dacă commitId nu există
- **-3: Vcs: There are staged operations, please do commit/rollback!**
  - dacă staging-ul nu este gol. Trebuie golit cu una din comenzile commit sau rollback

Comanda log

Afișează toate commit-urile date pe branch-ul curent.

Exemplu:

```
vcs log
```

Exemplu output

```
Commit id: 3
Message: First commit

Commit id: 5
Message: "Add list items functionality"
```

```
Commit id: 7
Message: "Optimize searching"
```

Comanda rollback

Golește staging-ul și aduce snapshot-ul de filesystem la versiunea dată de ultimul commit.

Exemplu:

```
1. vcs commit -m "Add changes"
2. touch a
3. ls
4. vcs status
5. vcs rollback
6. vcs status
7. ls
```

Exemplu output

```
3. a
   b
4. On branch: master
   Staged changes:
       Created file a
6. On branch: master
   Staged changes:
7. b
```

Recomandări

- Scheletul de cod este opțional. Asfel, el poate fi modificat sau înlocuit cu o implementare proprie. Dacă optați pentru o implementare proprie, trebuie să vă asigurați ca trec testele de filesystem.
- Pentru implmentarea commit-ului nu trebuie să vă axați strict pe eficiență. Puteți salva întreg sistemul de fișiere în cadrul unui commit, nu doar fișierele care au fost modificate.

Structură arhivă

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

- fișierul de tip **Makefile**, avand numele `VCSMakefile`, care sa includa regulile build și clean
- fișierul `Main.java` (entry-point-ul aplicatiei voastre care nu se află în vreun pachet)
- fișierele din scheletul temei
- alte fișiere **organizate** cu implementarea **voastră**
- fișierul README

Nu încărcați fișierele de test, checker-ul sau documente generate cu JavaDoc.

Nu incarcati fisierul poo_checks.xml. Versiunea acestui fisier este cea din checker.
Daca considerati ca anumite reguli din checkstyle trebuie schimbate, deschideti un issue pe pagina de GitHub a organizatiei.

Notare

- 90p teste publice (15 de teste a cate 6p fiecare)
- 10p README (care surprinde cele mai relevante detalii de implementare)

În cadrul arhivei sunt câteva teste care conțin numai comenzi de filesystem. Acestea nu vor fi punctate, dar reprezintă o bună modalitate de verificare a temei în momentul în care optați pentru o implementare proprie a scheletului.

Checkstyle-ul va efectua doar o verificare, dar nu va putea aduce un punctaj adițional. **Totuși, la peste 30 de erori rezultate din acesta, se va scădea câte 1p (din 100) pentru fiecare warning.**

Exemple punctare:

- 15 teste OK, README, 0 checkstyle warnings → 100p
- 15 teste OK, README, 30 checkstyle warnings → 100p
- 15 teste OK, README, 31 checkstyle warnings → 69p
- 15 teste OK, README, 55 checkstyle warnings → 45p

Nu uitați să citiți pagina Indicații teme.

Rularea temei pe **VMChecker** cu cele 15 teste trebuie să se încadreze în timpul de timeout de **120 de secunde**. Încercați astfel să nu lăsați pe ultima sută de metri rezolvarea, deoarece se pot forma cozi de așteptare pentru testarea pe **VMChecker**.

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

You shall indeed not pass!

Dacă optați pentru o implementare proprie a scheletului, asigurați-vă că trec testele de filesystem. Dacă unul din testele de filesystem pică, checkerul nu acordă punctaj pe temă.

Resurse

- Scheletul de cod: https://github.com/oop-pub/teme/tree/master/tema2
- Checkerul: https://github.com/oop-pub/teme/tree/master/tema2

Updates

- 24 noiembrie - precizare poo_checks.xml
- 10 noiembrie - explicatie suplimentara vcs checkout -c
- 10 noiembrie - corectare greseli test9.ref, test11.ref
  - test9: dupa ce dati un checkout -c X, sa stergeti commit-urile ulterioare commit-ului X
  - test11: rollback trebuie sa stearga modificarile doar pana la ultimul commit dat, adica pana la commit-ul “Add c and d files”
- 9 noiembrie - adăugarea listei de operații track-uite de vcs status
- 7 noiembrie - adăugare checker și fișiere descriptive

Referințe

- Tutorial checkstyle