POO - TEMA 3

Responabil tema: Georgiana Ciocirdel (georgiana.ciocirdel@stud.acs.upb.ro)
Data propunerii: 9 decembrie 2015
Deadline: 6 ianuarie 2016 (hard)

INTRODUCERE

Un grup de persoane nemultumite a cazut de comun acord ca Eclipse si IntelliJ nu sunt IDE-urile ideale pentru a construi aplicatii Java, asa ca a pornit un proiect open source pentru a scrie un editor corespunzator. Cand editorul a fost gata (cativa ani mai tarziu), cei implicati au decis sa sustina o serie de sesiuni de “usability testing”, dorind astfel sa vada impactul pe care il are schimbarea adusa de ei asupra utilizatorilor. Scopul studiului era acela de a determina daca aranjarea elementelor in editor (buton de run, fereastra de scris cod, consola de debugging, meniu, etc) sunt sau nu usor de gasit si accesat de catre utilizatori.

Acest lucru a presupus mai intai gasirea unor voluntari care sa cunoasca Java la diferite nivele. Intr-o sesiune de 30 minute, fiecarui voluntar i s-au dat de indeplinit cateva task-uri. La editor s-a adaugat un plug-in, care inregistra fiecare click dat de utilizator in IDE. Log-urile au fost scrise intr-un fisier separat pentru fiecare task si pentru fiecare utilizator, fiecare intrare in log fiind considerata un eveniment si fiind compusa din:

- o lista ce contine identificatori unici ai elementului pe care s-a dat click;
- timestamp-ul evenimentului.

Urmatorul pas este acela de a prelucra log-urile. Pentru simplificare (in general, intr-o sesiune de 30 de minute se pot strange sute de evenimente), sa consideram doar trei dintre task-urile ce i s-au dat unuia dintre utilizatori si sa trunchiem numarul de log-uri. Pentru ca identitatea utilizatorului nu trebuie dezvaluita, sa presupunem ca se numeste Alfred.

Analiza sesiunii presupune: parsarea task-urilor reprezentate de log-urile scrise in fisier, determinarea unor statistici bazate pe frecventa de click-uri si numarul de evenimente adunate, precum si gasirea unei metode de a produce un output grafic, care sa ii ajute pe cei ce interpreteaza log-urile sa inteleaga mai usor ce s-a intamplat in timpul sesiunilor.

In continuare, este prezentata structura proiectului primit, a fisierelor de input, precum si task-urile temei, care incearca sa rezolve cele trei probleme de mai sus. Ele nu au o ordine fixa, le puteti aborda cum doriti.

STRUCTURA PROIECTULUI

Ierarhia pachetelor si claselor

Este urmatoarea:

```
assignment/
     src/
        graphics/    -> pachetul grafic
        io/          -> pachetul pentru operatii I/O
        visualiser/  -> pachetul pentru parsarea task-urilor si
                        calculul statisticilor
```

Fisierele de input se regasesc sub:

```
assignment/
     logs/
        alfred/
```

si au numele `task1`, `task2` si `task3`.

Structura fisierului de input

Un fisier de input arata astfel:

```
user_event {
    element : "input"
    element : "div"
    element : "div"
    timestamp : 0
} user_event[0]
user_event {
    element : "card"
    element : "div"
    timestamp : 1
} user_event[1]
```

Asta inseamna ca au avut loc doua evenimente (click-uri), iar identificatoarele lor unice au fost cele doua secvente: “`input`”, “`div`”, “`div`”, respectiv “`card`”, “`div`”. Primul eveniment a avut loc la secunda `0`, iar cel de-al doilea la secunda `1`.

TASK 1 - Citire (5p)

Veti gasi logurile lui alfred in arhiva temei, in folderul `logs/alfred/`. Primul task este acela de a citi fisierele de log. Pentru a nu ne complica la pasul acesta, implementati functia `readFromFile` din clasa `Parser` (pachetul `io`), care citeste un fisier si intoarce un `String` din continutul acestuia.

Continutul fisierului (cum apare mai sus) trebuie sa fie identic cu ceea ce intoarce functia `readFromFile(String)`.

TASK 2 - Parsare (45p)

In functia `readTasks` in clasa `Main` creati o lista de `Task`uri si adaugati-i, pe rand, dupa fiecare citire de fisier (cu functia de la task-ul 1), cate o instanta. Clasa `Task` se regaseste in pachetul `visualiser` si trebuie sa contina urmatoarele campuri si metode:

- o lista de evenimente (descrierea clasei `UserEvent` se regaseste mai jos);

```
private final List<UserEvent> userEvents;
```

- un constructor fara argumente;
- un constructor ce primeste ca argumente un obiect `String` - acesta reprezinta log-urile citite dintr-un fisier (tot continutul fisierului);
- o functie care parseaza logurile primite si intoarce o lista de evenimente. Ganditi-va unde trebuie sa apelati aceasta functie.

```
List<UserEvent> parseLogs(String logs)
```

- o functie care primeste ca argument o lista de elemente ce identifica in mod unic o zona din pagina unde s-a dat click si intoarce o instanta a zonei respective (clasele `PageArea` si `PageElement` sunt descrise mai jos).

```
EditorArea determineAreaForElements(List<EditorElement> elements)
```

- o functie care calculeaza frecventa medie de click-uri pe intervale de 10s.

```
public Double meanFrequencyPerTenSeconds()
```

Exemplu:
Sa presupunem ca avem urmatoarele timestamp-uri consecutive in fisierul de log-uri:
`0 1 5 15 16 19 21 30`
Intervalele de 10 secunde sunt intre urmatoarele timestamp-uri:
`[0, 9], [10, 19], [20, 29], [30, 39]`
Cele 8 evenimente se mapeaza astfel:

```
[ 0,  9] <- 0, 1, 5
[10, 19] <- 15, 16, 19
[20, 29] <- 21
[30, 39] <- 30
```

Astfel, frecventele pe intervale sunt:

```
[ 0,  9] <- 3 / 10 = 0.3
[10, 19] <- 3 / 10 = 0.3
[20, 29] <- 1 / 10 = 0.1
[30, 39] <- 1 / 10 = 0.1
```

Iar frecventa medie va fi:
`(0.3 + 0.3 + 0.1 + 0.1) / 4 = 0.8 / 4 = 0.2`

ATENTIE: Nu intotdeauna primul task va avea timestamp-ul de inceput `0` ! Din moment ce sunt logate 3 task-uri consecutive, task-urile 2 si 3 vor avea primul timestamp mai mare decat cel al primului task, iar task-ul 3 va avea timestamp-ul mai mare decat cel al task-ului 2.

- o functie care va calcula numarul de click-uri care s-au dat pe fiecare zona in parte, pentru fiecare zona distincta din fisierul de log-uri. Functia intoarce o mapare intre cheia reprezentand numele clasei care abstractizeaza zona (se regasesc mai jos detalii despre fiecare astfel de clasa) si numarul de click-uri date.

```
public Map<String, Double> computeClicksPerArea()
```

HINT: Pentru acest task, am atasat fisierul de test `TestTask.java`; observati testele `clicksPerArea_*`. La fiecare inceput de test, se construieste outputul care se asteapta sa fie produs de functia testata. Observati ce se foloseste pentru cheia maparii in fiecare test.

TASK 3 - Completarea resurselor necesare, EditorArea (6p)

In acest task, va trebui sa implementati clasele auxiliare, care va vor ajuta pe parcursul temei. Toate clasele se regasesc in pachetul `visualiser`.

`UserEvent`

- descrie un eveniment care a avut loc in editor;
- contine campurile:

```
private PageArea area;
private int timestamp;
```

  care desemneaza zona din pagina unde s-a dat click si timestamp-ul evenimentului.

- constructori:

```
public UserEvent()
public UserEvent(PageArea area, int timestamp)
```

- metode:

```
setteri / getteri
```

`EditorElement`

- abstractizeaza un element al IDE-ului;
- contine campurile:

```
private final String type;
```

- constructori:

```
public EditorElement(String type)
```

- metode:

`getteri / setteri` (Intrebare: Se pot implementa setteri pentru clasa aceasta ? De ce ?)

`EditorArea (abstract)`

- clasa abstracta, care descrie o zona din editor;
- contine campurile:

`private final List<PageElement> pathInEditor;` <- O lista de elemente care identifica in mod unic un element al editorului.

- constructori:

`public EditorArea(List<EditorElement> pathInEditor)`

- metode:

`public abstract Color getVisualisationColor()` (despre care vom vorbi mai jos)
`getteri / setteri` (Intrebare: Se pot implementa setteri pentru clasa aceasta ? De ce ?)

Clasa `EditorArea` va fi extinsta de urmatoarele clase:
`Canvas`
`DialogBox`
`Menu`
`UnknownArea`

Fiecare din aceste clase va trebui sa implementeze metoda `getVisualisationColor()` si sa intoarca o culoare diferita. Culorile se vor declara intr-o enumeratie, `Color`. Declarati cate o culoare pentru fiecare clasa (ex: `RED, GREEN, ...`). Puteti defini orice culoare doriti, dar pastrati GREY pentru `UnknownArea` si pentru task-ul de vizualizare (4).

In momentul parsarii log-urilor, se va determina ce zona din editor reprezinta fiecare eveniment, pe baza urmatorului filtru:

- daca log-urile contin cuvintele cheie `card, input` sau `page` => `Canvas`;
- daca log-urile contin cuvintele cheie `menu, menu-button sau icon` => `Menu`;
- daca log-urile contin cuvantul cheie `dialog` => `Dialog`;
- altfel => `UnknownArea`.

(6p) Utilizati un design pattern adecvat pentru a “fabrica” cele patru zone de editor posibile, in functie de log-uri. Adaugati orice clasa considerati a fi necesara.

TASK 4 - Clustering (22p)

Dupa parsarea log-urilor, dorim sa vedem si sa masuram rezultatele. Inainte de orice, insa, trebuie sa ne gandim cum arata un utilizator dezorientat in log-uri.

Ca sa incercati un exemplu concret de ce ar insemna prima experienta cu un IDE (si, foarte probabil, un limbaj complet nou), incercati Dartpad, un editor online de Dart si rezolvati urmatoarele task-uri:

1) Pad-ul ofera niste exemple de programe scrise in Dart. Ce numar este folosit in exemplul sirului lui Fibonacci ?
2) Cand anume devine butonul de Reset utilizabil ?

3) Ati putea sa preluati cumva pad-ul curent si sa il folositi intr-o pagina web, ca de exemplu aici ?

Daca ati incercat sa rezolvati task-urile de mai sus, ganditi-va pe ce butoane ati apasat, cat de mult v-a luat sa rezolvati fiecare task, cat “v-ati plimbat in pagina”.

Un alt exemplu, din lumea site-urilor: Intrati pe site-ul Comisiei Europene. Cand a fost ultima oara updatata sectiunea unde ni se spune cum sunt tratate in mod legal email-urile trimise catre Comisie ?

In general (poate nu a fost cazul vostru), un utilizator care este “pierdut” se plimba printre aceleasi elemente ale paginii, dand click-uri rapide pe un buton, apoi pe altul, revenind de unde a plecat, cautand mereu solutia, dar fara insa a realiza un progres.

Printre cele 3 task-uri care i s-au dat lui Alfred, se afla unul pe care a reusit sa il rezolve rapid, dar la celelalte doua a ezitat mult. Cum am putea vedea asta prin calcule ?

In primul rand, putem calcula frecventa click-urilor pe intervale fixate, cum am facut la Task-ul 2, pentru intervale de 10 secunde. Apoi, tot la task-ul 2, am numarat click-urile date pe fiecare zona a IDE-ului in parte. Cu toate acestea, o frecventa mare a click-urilor nu inseamna neaparat cu un utilizator era dezorientat - poate inseamna doar ca stia exact ceea ce face si a rezolvat task-ul cu succes.

TASK 4.1 (4p)

Care zona a editorului pare sa fi fost utilizata cel mai mult de Alfred ? (numara cele mai multe click-uri)
Care task ***pare*** a fi cel mai problematic (are frecventa mare a click-urilor) ?

TASK 4.2 (12p)

Inainte de a vizualiza rezultatele (sub o forma sau alta) va trebui sa determinam zonele care ***par*** problematice, adica acele zone unde s-a dat click in mod repetat si la distanta scurta de timp. Pentru acest lucru, trebuie implementata clasa `ClusterManager`:

- campuri:

`private static int window = 10` <- va determina durata de timp pe care vom cauta elemente repetitive, relativ la timestamp-ul primului element al clusterului (nu e necesar sa va definiti acest camp, puteti inlocui direct cu valoarea 10);

- constructori:

```
public ClusterManager(int window)
```

- metode:

```
public List<Cluster> cluster(List<UserEvent> userEvents)
```

- clasa interna `Cluster`, cu campurile si metodele aferente:
  - campuri:

```
private List<UserEvent> userEvents;
private int startTimestamp;
```

`private int endTimestamp;`

- constructor:

`public Cluster(List<UserEvent> userEvents, int start, int end)`

- metode:

`public void addUserEvent(UserEvent e)` <- adauga un nou eveniment la acest cluster

`setteri / getteri`

Algoritmul de clustering trebuie sa identifice secventele repetitive ce contin acelasi element al editorului, incadrate de fereastra `window`. Exemplu:

`e1, e2` = doua zone diferite ale editorului, `window = 10`

`[e1, t1 = 0], [e2, t2 = 5], [e1, t3 = 8], [e1, t4 = 9]`

`window = 10`

Atunci, obtinem clusterele:

`[[e1, t1 = 0], [e1, t3 = 8], [e1, t4 = 9], start = 0, end = 19]`

`[[e2, t2 = 5], start = 5, end = 15]`

Observati ca timpul de final devine mereu timestamp-ul ultimului eveniment adaugat + fereastra - 1.

PRECIZARE: Secventele cu un eveniment vor fi ignorate din rezultatul final.

ATENTIE: Pe langa metodele descrise pentru fiecare clasa de mai sus, va trebui sa implementati metoda `equals()`, `hashCode()` (si, eventual `toString()`, daca vreti sa printati vreun rezultat intermediar) pentru fiecare clasa in parte. Daca nu implementati aceste metode (`equals` si `hash`), testele nu vor trece :)

De asemenea, orice implementare de tipul `return true;` a metodei `equals()` atrage dupa sine anularea completa a punctajului pe teste.

TASK 4.3 (6p)

Utilizati `Singleton Pattern` pentru a implementa `ClusterManager`-ul.

TASK 5 - Vizualizare (10p)

Mai avem de facut cativa pasi pana a putea vedea rezultatele studiului.

TASK 5.1 (4p)

In pachetul `graphics` gasiti clasa `Point.java` care va retine detaliile unui punct pe graficul final. Un punct va trebui definit prin pozitia sa pe axele `x` si `y`, precum si prin culoarea sa (asa cum este definita in enumeratia `Color`. Implementati clasa `Point` cu ajutorul `Fluent Builder Pattern`.

TASK 5.2 (8p)

In pachetul `graphics` gasiti clasa `Renderer.java` care se va ocupa de desenarea evenimentelor din log-uri. Un `Renderer` va avea nevoie de o lista de puncte (pe care le va desena), dimensiunea unui punct (`circleSize`), precum si de un mod de a face legatura intre o culoare definita in enumeratia `Color` si o culoare din pachetul `awt`, pe care o va folosi pentru desenarea punctelor. `Renderer`-ul este o clasa derivata din clasa `JFrame`, asa ca puteti seta o metoda de inchidere a ferestrei pe care se va desena. Un exemplu ar fi `JFrame.EXIT_ON_CLOSE` (vezi `main`). De asemenea, `Renderer`-ului i se poate seta si un titlu. Folositi `Fluent Builder Pattern` pentru a implementa clasa si a ii conferi toate aceste proprietati.

TASK 5.3 (1p)

Adaugati in `main` o mapare pentru fiecare culoare definita in enumeratia `Color` catre o culoare din biblioteca grafica `awt`, construita prin valori RGB. De exemplu, pentru rosu, maparea ar arata astfel:

```
visualiser.Color.RED -> new Color(255, 0, 0) (am folosit codul RGB)
```

Puteti, astfel, folosi orice culoare doriti. Aveti grija, totusi, sa utilizati culori cat de cat diferite una de cealalta, pentru a putea diferentia intre elementele editorului. Notati in README culorile folosite.

TASK 5.4 (5p)

Folosindu-va de metoda `draw` din clasa `Renderer`, desenati, mai intai toate evenimentele inregistrate in cele 3 task-uri. Apoi, desenati din nou toate evenimentele, cu gri, de data aceasta si apoi desenati fiecare cluster in parte cu o culoare **distincta** (astfel incat, uitandu-va la rezultat, sa puteti diferentia intre clustere).

Pentru a desena, va trebui initial sa decideti asupra dimensiunii frame-ului (in exemplul de mai jos `1500 / 300`). Fiecare eveniment va fi reprezentat de un cerc, colorat diferit in functie de zona in care s-a dat click. Dimensiunea cercului, in cazul de mai jos este `10`. Folositi un titlu semnificativ pentru fiecare grafic. In exemplul de mai jos, am folosit `"All data"` si `“Clusters”`.

Gasiti o formula potrivita pentru a calcula coordonatele `x` si `y` ale evenimentelor. In imaginile de mai jos, spre exemplu, am folosit timestamp-ul unui eveniment pentru `x`, iar pentru `y` am folosit un numar unic, determinat incremental, pentru fiecare lista de elemente ce identifica in mod unic un eveniment. De exemplu, am mapat `card` cu numarul 1, `input` cu 2, etc. Astfel, toate evenimentele care au avut loc asupra aceleiasi zone (aproximativ) din editor, vor ajunge pe aceeasi axa `y`. Desigur, puteti alege orice alta reprezentare doriti.

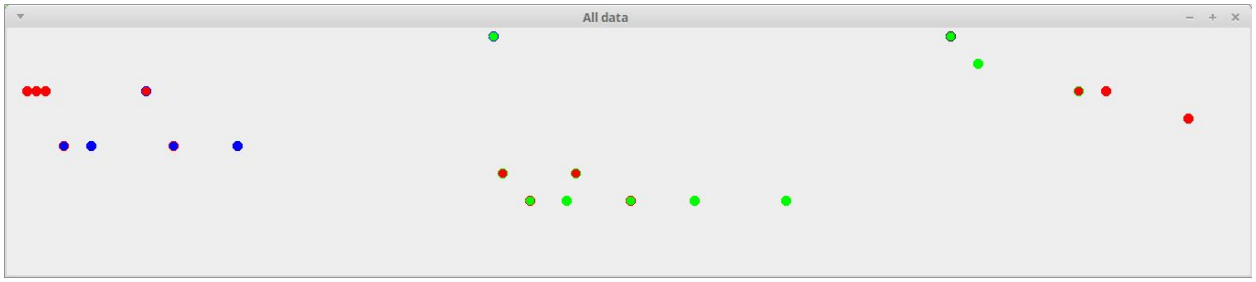
Implementati metodele `drawAll` si `drawClusters` in clasa `Main`. In primul desen adaugati toate evenimentele in grafic, colorand punctele diferit in functie de zona pe care s-a dat click. Pentru desenarea clusterelor, adaugati toate punctele (din toate task-urile) cu gri (sau o culoare similara), dupa care desenati clusterele obtinute. Desenati fiecare cluster cu o culoare diferita de celelalte, ca sa se poata identifica usor. Puteti adauga noi culori in clasa `Color` sau puteti folosi culorile deja existente. Alegeti o dimensiune a ferestrei care vi se pare potrivita (care surprinde cat mai bine secventele de elemente ce se repeta).

Notati in README fereastra folosita la clustering, dimensiunile folosite pentru frame, precum si formulele folosite pentru a determina valorile de pe axele `x` si `y`. De asemenea, descrieti algoritmul de clustering folosit.

Pe niste fisiere de log similare cu cele atasate arhivei cu tema, graficele ar putea arata ca mai jos. Am folosit urmatorii parametrii (si formule pentru x si y, astfel incat toate punctele sa fie vizibile in frame si sa se poata distinge unul de celalalt):

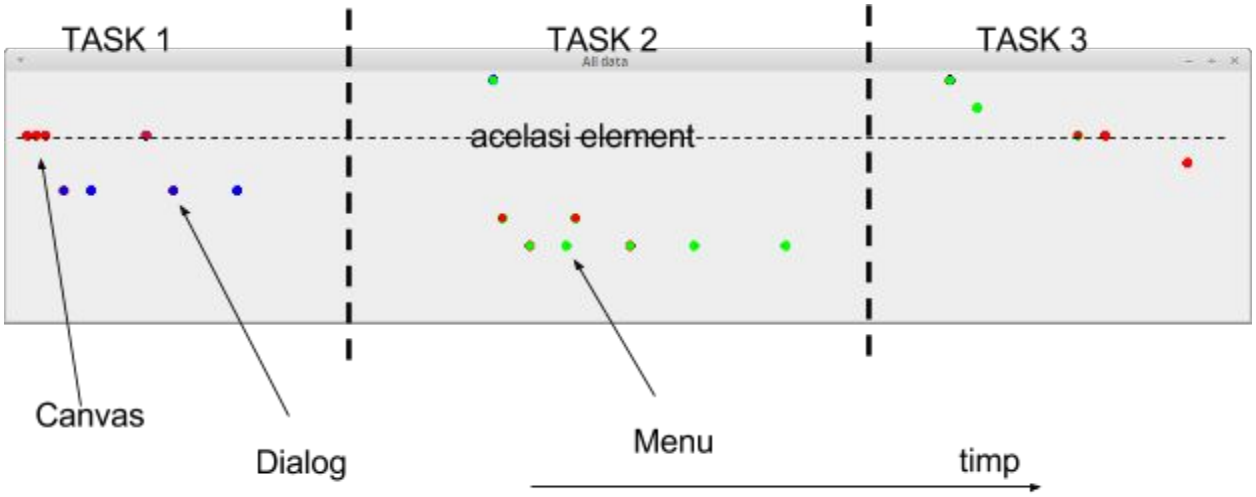
```
Canvas  =  RED
Menu  = GREEN
DialogBox  = BLUE
window = 10
```

Toate datele din fisiere:



Clusterele (am reprezentat restul de puncte cu alb):

Ce vedem in imaginile de mai sus ?
Cele trei task-uri se disting, pentru ca exista o pauza de timp intre ele.



La task-ul 1, Alfred a fost un pic dezorientat. A dat click-uri repetate in meniu, dupa care a deschis un dialog box, s-a intors la meniu, dupa care la dialog box.
La task-ul 2, situatia pare a fi similara, Alfred, plimbandu-se intre meniu si pagina editorului.
Observati ca si algoritmul de clustering identifica niste zone “problematice” in cadrul acestor task-uri.
La task-ul 3, Alfred pare sa fi rezolvat, totusi, fara probleme task-ul. Desi algoritmul de clustering identifica o zona cu doua evenimente foarte apropiate, acest lucru poate insemna ca Alfred stia unde sa dea click si s-a deplasat rapid printre elementele editorului.

TASK 5.5 (4p)

Atasati in fisierul README un screenshot cu graficul obtinut de voi pentru toate evenimentele si unul cu graficul obtinut pentru clustering.
Ce puteti spune despre fiecare task in parte ? Cum vi se pare ca au decurs cele 3 task-uri pentru Alfred ?

TESTARE

Testarea se va face automat pentru clasele `Task` si `ClusterManager`. Testele se regasesc in clasele `TestTask.java` (15 teste) si `TestClusterManager.java` (4 teste) si ruleaza cu `JUnit4`. Pentru a putea rula testele, veti avea nevoie sa adaugati biblioteca `Guava` in buildpath-ul proiectului vostru. Jar-ul necesar este inclus in arhiva temei, in folder-ul `lib/`. Detalii pentru Eclipse si IntelliJ. Instructiuni simple de rulare a testelor in Eclipse si IntelliJ. Pentru a face debugging unui test, puteti folosi orice tool de debugging disponibil in IDE (ex pentru Eclipse), dar, daca va este mai usor sa folositi “printare la consola” puteti face acest lucru in orice metoda de test doriti.

Veti observa ca unele metode au fost anotate cu `@VisibleForTesting` si au vizibilitate package-protected. Acest lucru inseamna ca ele devin vizibile pentru testele care se afla in interiorul aceluiasi pachet.

Instructiuni pentru rularea in linie de comanda a checker-ul adaugat temei:
In folderul `assignment/` am adaugat un fisier `Makefile`, care compileaza toate sursele si apoi ruleaza cele doua fisiere de test. In folderul `lib/` am inclus bibliotecile necesare pentru rularea acestor teste. Comanda pentru rularea testelor este:

```
gigi@terminal ~/path/to/assignment$ make test
```

Task-urile care necesita raspunsuri in README vor fi punctate separat.

ARHIVA PRIMITA

Ce se regaseste in arhiva temei ?

- folderul `assignment` cu pachetele, clasele si fisierele de log si fisierul `Makefile` mentionat mai sus;
- un fisier TEXT README care contine intrebarile la care trebuie sa raspundeti (au fost notate pe parcursul enuntului si centralizate in acest fisier);
- un folder `lib/` cu jar-ul bibliotecii Guava si alte biblioteci necesare rularii testelor.

ARHIVA DE TRIMIS

Ce trebuie sa trimiteti ?

- folderul `assignment` cu implementarile necesare in folder `src/`, folderele `lib/`, `bin/` si `logs/`, precum si fisierul `Makefile` nu sunt necesare la upload-ul temei;
- nu este nevoie sa atasati si testele, acestea vor fi scrise/suprascrise in timpul corectarii;
- nu este nevoie sa trimiteti fisierele de log;
- un fisier README in format **PDF**, in care ar trebui sa:
    - sa va mentionati numele si grupa ! (temelor nesemnate nu li se vor cauta proprietarii, ci vor ramane fara punctaj)
    - raspundeti la intrebarile scrise in fisierul TEXT intial;
    - atasati screenshot-urile cerute.
- asigurati-va ca README-ul este in format **PDF**. Nu vom lua in considerare alte formate !

PUNCTAJUL - REZUMAT

TASK 1	5p	cod
TASK 2	45p	15 teste, 3p/test
TASK 3	6p	cod
TASK 4.1	4p	README
TASK 4.2	12p	4 teste, 3p/test
TASK 4.3	6p	cod
TASK 5.1	4p	cod
TASK 5.2	8p	cod
TASK 5.3	1p	README + cod
TASK 5.4	5p	README + cod
TASK 5.5	4p	README
CODING STYLE	10p	
TOTAL	110p	

CHANGELOG

Task2: `**Page**Area determineAreaForElements(List<**Page**Element> elements) -> **Editor**Area determineAreaForElements(List<**Editor**Element> elements)`

Task3: `public **Page**Area(List<**Page**Element> pathInPage) -> public **Editor**Area(List<**Editor**Element> pathInEditor)`

Task4.2: `private final int window` <- va determina durata de timp pe care vom cauta elemente repetitive, relativ la timestamp-ul primului element al clusterului -> `private **static** int window = 10` <- va determina durata de timp pe care vom cauta elemente repetitive, relativ la timestamp-ul primului element al clusterului **(nu e necesar sa va definiti acest camp, puteti inlocui direct cu valoarea 10)**;

Task 4.2: Observati ca timpul de final devine mereu timestamp-ul ultimului eveniment adaugat + fereastra. -> Observati ca timpul de final devine mereu timestamp-ul ultimului eveniment adaugat + fereastra **- 1**.

Responsabil tema: Georgiana Ciocirdel (georgiana.ciocirdel@**cti.pub.ro**) -> Responsabil tema: Georgiana Ciocirdel (georgiana.ciocirdel@**stud.acs.upb.ro**)