








Proiect - Etapa 1 - Sistem energetic

- **Data publicării:** 29.11.2020 20:00
- **Data ultimei modificări:** 01.12.2020 14:20
- **Deadline soft:** 15.12.2020 23:55
- **Deadline hard:** 16.12.2020 23:55 20.12.2020 23:55
- **Responsabili:**  Ionuț Bîrsu,  Raimond Varga ,  Bianca-Andreea Ciuche,  Bogdan Fîruți,  Narcis-Florin Căroi,  Andreea Oltean,  Anca Enache

Obiective

- dezvoltarea unor abilități de bază de organizare și design orientat-obiect
- scrierea unui cod cât mai generic, ce va permite ulterior adăugarea de noi funcționalități
- folosirea unor design patterns
- respectarea unui stil de codare și de comentare

Scenariu

Rețeaua de curent electric a unei țări este un sistem complex, format din producători, distribuitori, consumatori (casnici sau industriali) și instituții ale statului care reglementează și supraveghează buna funcționare a sistemului.

Proiectul se bazează pe simularea unui sistem energetic în care vom avea diferite entități cu atribuții bine definite, care vor fi introduse pe parcurs - producători, distribuitori, consumatori, etc. Toate aceste entități încearcă să-și îndeplinească îndatoririle având drept scop final rămânerea pe plată și evitarea falimentului. Acesta este structurat pe două etape, deci este recomandat să aveți un cod cât mai generic, pentru a fi ușor de adăugat noi funcționalități la etapa următoare. În aceasta etapă vom lua în considerare 2 entități: distribuitori și consumatori.

Consumatori

- Doresc o sursă de energie;
- Au un buget inițial + venit lunar;
- Aleg contractul cu cea mai mică rată lunară;
- Plătesc rată stabilită la început pentru o perioadă stabilită la alegerea contractului.
- În cazul în care acesta rămâne fără bani, el poate amâna o luna plata facturii (se poate amâna inclusiv plata primei facturi din contract), cu singura condiție ca în următoarea lună să plătească factura pe luna curentă și pe luna trecută, dar și o penalizare egală cu 20% din valoarea facturii neplătite.
- Dacă nici luna viitoare nu poate plăti spre același distribuitor, atunci acesta va declara faliment, fiind exclus din joc. Modul de plată în cazul unei penalizări este:

$$\text{Math.round}(\text{Math.floor}(1.2 * \text{factura veche})) + \text{factura nouă}$$
- Consumatorul va putea căuta un nou distribuitor chiar dacă are de plătit o factură veche cu penalizare spre distribuitorul vechi.
- Un consumator este considerat clientul unui distribuitor până își alege alt contract sau până la finalul lunii în care dă faliment.

Distribuitori

Distribuitorii vor reprezenta companiile responsabile cu oferirea surselor de energie. Fiecare astfel de companie va fi vizitată în fiecare luna de consumatori dornici să afle ce oferta pot obține. Un distribuitor poate schimba caracteristicile noulor contracte (costurile de întreținere și durata) în fiecare luna, noile valori regasindu-se în input. Distribuitorii vor avea și ei un buget de început, la care lunar se va adăuga profitul obținut. Întrucât aceștia au de achitat prețuri atât pentru infrastructură cât și pentru producție, formula prețului contractului diferă în funcție de numărul de clienți.

Prețul contractului va fi recalculat la începutul fiecărei luni folosind noile informații despre distribuitor primite în input. Costurile vor fi precizate în fișierul de intrare iar profitul va fi considerat 20% din costul de producție. Un distribuitor, indiferent dacă are sau nu clienți, va fi nevoit ca la finalul fiecărei luni să plătească costurile pentru infrastructură companiei. Costul de producție va fi plătit doar dacă exista clienți la finalul acelei luni. Astfel, dacă o companie nu reușește să atragă clienți pe un timp îndelungat, atunci aceasta va da faliment și va ieși din joc. Clienții firmei ce a dat faliment își vor căuta o nouă oferta luna viitoare.

Mecanismul simulării

Simularea este bazată pe runde reprezentate de luni. La finalul fiecărei luni, toți consumatorii înregistrați trebuie să aibă o sursă de electricitate, altfel se consideră că aceștia nu își pot permite și sunt dați afara din joc. La fel se întâmplă și cu distribuitorii ce rămân fără bani, cu precizarea că în momentul falimentării, toți consumatorii ce au contract cu acesta, vor trebui să își formeze alt contract începând cu luna următoare.

La începutul fiecărei luni distribuitorii vor stabili noile prețuri, urmând ca după ce aceștia termină, să vină rândul consumatorilor, care vor alege un contract (dacă nu au deja unul), urmând ca la finalul lunii să plătească rata curentă. Simularea începe cu o rundă inițială, unde sunt folosite datele primite la început, apoi sunt rulate numberOfTurns luni, care se folosesc de noile prețuri primite la începutul fiecărei luni. Astfel, simularea se termină când au fost rulate numberOfTurns + 1 runde și se afișează starea curentă a simulării. În cazul în care toți distribuitorii dau faliment, jocul se va încheia.

Formulele:

- Prețul final al contractului:

$$\text{Math.round}(\text{Math.floor}(\text{costul infrastructurii} / \text{numărul de consumatori curenți})) + \text{costul producției} +$$
- Prețul final al contractului dacă nu exista clienți:

$$\text{costul infrastructurii} + \text{costul producției} + \text{profitul}$$
- Profitul:

$$\text{profit} = \text{Math.round}(\text{Math.floor}(0.2 * \text{costul producției}))$$
- Cheltuieli lunare distribuitori:

$$\text{Cost total} = \text{cost infrastructură} + \text{cost producție} * \text{număr de consumatori curenți}$$

Cerinte

Ne dorim să modelăm această simulare în stilul orientat-obiect. Vom citi configurația și desfășurarea unui joc dintr-un fișier de intrare, vom rula jocul și vom scrie într-un fișier de ieșire stările entităților. În soluțiile voastre, entry-point-ul (**metoda public static void main(String[] args)**) va fi clasa numită **Main**, aflată în scheletul temei. Primul argument

 Search

Administrativ

- Regulament
- Echipa
- Orar
- Indicații pentru activitățile online
- Recomandări cod
- Indicații pentru teme

Cursuri

- Cursuri seria CA
- Cursuri seria CD

Laboratoare

- Lab 01 - Java Basics
- Lab 02 - Constructori și referințe
- Lab 03 - Agregare și moștenire
- Lab 04 - Static, Final, Singleton
- Lab 05 - Abstractizare
- Lab 06 - Clase interne
- Lab 07 - Overriding, Overloading & Visitor pattern
- Lab 08 - Colecții
- Lab 09 - Design Patterns
- Lab 10 - Design Patterns
- Lab 11 - Genericitate
- Lab 12 - Java features
- Lab 13 - Excepții

Teme

- Tema - VideoșDB
- Etapa 1 - Sistem energetic
- Etapa 2 - Sistem energetic

Teste

- Teste grilă

Resurse utile

- Instalare IntelliJ Idea
- Activare IntelliJ Idea
- Demo proiect IntelliJ Idea
- Tutorial Git in IntelliJ Idea
- Tutorial checkstyle

Alte resurse

- Laborator recapitulare
- Exerciții vechi
- POO și Java
- JUnit
- Organizarea surselor și controlul accesului
- Tutorial I/O
- JSON & Jackson
- Double Dispatch - scurt tutorial
- Reflection

Arhiva Teme

- 2019-2020
- 2018-2019
- 2017-2018
- 2016-2017
- 2015-2016
- 2014-2015
- 2013-2014

Table of Contents

- Proiect - Etapa 1 - Sistem energetic
- Obiective
- Scenariu
 - Consumatori
 - Distribuitori
- Mecanismul simulării
 - Formulele:
- Cerinte
 - Input
 - Output
- Indicații
 - Testarea soluției
 - Evaluare
 - Checkstyle
 - Upload temă
 - Resurse și linkuri utile

este numele fișierului de intrare, al doilea este numele fișierului de ieșire. Nu schimbați numele clasei Main sau ordinea argumentelor. Implementarea voastră va trebui să conțină în mod obligatoriu design pattern-urile Factory și Singleton. De asemenea, în README, trebuie să documentați în care parte/părți din implementare l-ați folosit și să explicați, într-un mod clar și concis, modul de abordare. În explicații să vă axați pe flow-ul programului și pe modul în care se leaga componentele.

Input

Atât inputul, cât și outputul vor fi de tip json. Mai multe informații despre citirea fișierelor json [aici](#).

Click pentru exemplu input ↗

Output

Click pentru exemplu output ↗

Indicații

- Separați conceptele de sine stătătoare în clase separate, nu le îmbinați - clasele ar trebui să aibă un singur rol
- Adaptați agregarea și moștenirea la situație, grupați pe cât posibil informația și acțiunile comune în clase generale
- Nu vă apucați să scrieți direct; alocați timp modelării și abstractizării, pentru că altfel vă puteți trezi cu o temă muncitorească, cu mult cod din care să nu înțelegeți prea multe și pe care să îl extindeți greu
- Acesta este prima etapă a proiectului, ceea ce presupune că va exista și o a doua etapă; vă recomandăm să încercați să scrieți un cod cât mai generic, care să permită adăugarea ulterioară de noi funcționalități; cu toate acestea, etapa a doua se poate trimite fără să fi trimis prima etapă a proiectului, însă va fi nevoie ca în cadrul celei de-a doua etape să se implementeze și funcționalitățile primei etape pentru a putea primi punctajul total pe testele celei de-a doua părți
- Pentru calculele ce includ procente și numere cu zecimale, veți folosi doar partea întreagă a numărului.

Testarea soluției

Pentru testarea soluției, rulați funcția **main** a clasei **Test**. Aceasta va rula atât testele, cât și checkstyle-ul. Pentru rularea checkerului, aveți nevoie ca proiectul vostru să aibă încărcate bibliotecile pentru citirea fișierelor json. Mai multe detalii [aici](#).

Evaluare

Punctajul constă din:

- 90p implementare - trecerea testelor
- 10p coding style (vezi checkstyle)
- 15p design și organizare (folosire design patterns)
- 10p README clar, concis, explicații axate pe design (flow, interacțiuni)



Pe pagina [Indicații pentru teme](#) găsiți indicații despre scrierea readme-ului și depunctările generale pentru teme

Depunctările pentru **designul și organizarea codului** se vor scădea din punctajul testelor. Dacă vor apărea depunctări specifice temei în momentul evaluării, nemenționate pe pagina cu depunctări generale, ele se vor încadra în limitele de maxim 15 pentru design, 10p pentru readme. Dacă tema nu respecta cerințele, sau are zero design OOP atunci se pot face depunctări suplimentare.

Bonusuri: La evaluare, putem oferi bonusuri pentru design foarte bun, cod bine documentat dar și pentru diverse elemente suplimentare alese de voi.



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.

Checkstyle

Unul din obiectivele temei este învățarea respectării code-style-ului limbajului pe care îl folosiți. Aceste convenții (de exemplu cum numiți fișierele, clasele, variabilele, cum indentați) sunt verificate pentru temă de către tool-ul checkstyle.

Pe pagina de [Recomandări cod](#) găsiți câteva exemple de coding style.

Dacă numărul de erori depistate de checkstyle depășește 30, atunci punctele pentru coding-style nu vor fi acordate. Dacă punctajul este negativ, acesta se *trunchiază la 0*.

Exemple:

- `punctaj_total = 125 și nr_erori = 200 ⇒ nota_finala = 115`
- `punctaj_total = 125 și nr_erori = 29 ⇒ nota_finala = 125`
- `punctaj_total = 80 și nr_erori = 30 ⇒ nota_finala = 80`
- `punctaj_total = 80 și nr_erori = 31 ⇒ nota_finala = 70`

Upload temă

Arhiva pe care o veți urca pe VMChecker va trebui să conțină în directorul rădăcină:

- fișierul **README**
- folder-ul src cu pachetele și cu fișierele .java

Resurse și linkuri utile

- Schelet de cod
- [Indicații pentru teme](#)
- [Recomandări coding style & javadoc](#)

poo-ca-cd/teme/proiect/etapa1.txt - Last modified: 2020/12/16 13:18 by adriana.draghici

Old revisions

Media Manager Back to top