

Project - Etapa 2 - Sistem energetic

- **Data publicarii:** 30.12.2020
- **Data ultimei modificări:** 06.01.2021
- **Deadline hard:** 24.01.2021
- **Responsabili:** [Raimond Varga](#), [Adriana Drăghici](#), [Laurențiu Stamate](#), [Ionuț Bîrsu](#)
- **Repository:** [Github](#)

Obiective

- dezvoltarea unor abilități de bază de organizare și design orientat-obiect
- scrierea de cod pornind de la un alt cod existent, prin adăugarea de noi funcționalități
- crearea unui design decuplat și ușor ciblebil
- 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.

Etapa trecută s-a bazat pe interacțiunea dintre două entități - consumatori și distribuitori. În cadrul acestei etape, vom mai adăuga o entitate - producătorii - și vom extinde responsabilitățile distribuitorilor, păstrând în același timp și funcționalitățile de bază din prima etapă.

Consumatori

Aceștia funcționează ca în etapa trecută, cu o singura precizare suplimentară:

În cazul în care un consumator ramane cu o datorie în ultima lună de contract, acesta va fi în una din două situații:

- dacă nou contract este la același distribuitor, va trebui să plătească factura veche + penalități + factura nouă;
- dacă nou contract este la alt distribuitor, acesta poate plăti doar factura veche + penalități, amânând noua factură.

Distribuitori

În această etapă, distribuitorii vor interacționa și cu producătorii. În ceea ce privește consumatorii, interacțiunea va fi cea de la etapa anterioară, singura modificare fiind dată de înlocuirea costului de producție cu un cost obținut din cumpărarea energiei de la producători.

Interacțiunea cu producători:

- Distribuitorii au nevoie de o cantitate de energie lunară. Aceasta este constantă și dată ca input.
- În prima rundă distribuitorii își aleg unul sau mai mulți producători care să le ofere cantitatea de energie necesară
 - alegerea se face pe baza unor strategii - fiecare distribuitor are o strategie primită ca input
- Dacă a apărut o schimbare a cantității de energie oferită de un producător, atunci la începutul următoarei luni, distribuitorii care iau energie de la acel producător își vor aplica din nou strategia de alegere producători.

Schimbări lunare date de simulare:

- costul pentru infrastructură

În input și în schimbări lunare s-a eliminat costul de producție pentru că acesta va fi calculat pe baza energiei luate de la producători.

Producători

Aceasta reprezintă noua entitate introdusă în această etapă. Un producător va produce energie de un anumit tip și o va vinde mai multor distribuitori.

Proprietățile unui producător:

- tip de energie produsă, care poate fi regenerabilă (wind, solar, hydro) sau nu (coal, nuclear)
 - tipurile sunt definite într-un enum `EnergyType` dat în schelet
- preț per KWh
- cantitatea lunară de energie oferită fiecărui distribuitor - oferă aceeași cantitate de energie fiecărui
- numărul maxim de distribuitori către care poate oferi energie

Schimbări lunare date de simulare:

- energia oferită fiecărui distribuitor

Condiții, simplificări:

- Un producător nu este obligat să își vândă lunar energie către numărul maxim de distribuitori.
- Un distribuitor trebuie să își ia de la producători toată cantitatea de energie lunară necesară
- Vom considera că vor fi suficienți producători și cu cantități lunare oferite suficiente astfel încât să se poată acoperi tot necesarul distribuitorilor - testele vor asigura aceasta condiție

Strategii de alegere producători

- **Green Strategy** - un distribuitor își alege producătorii prioritizându-i pe cei cu renewable energy întâi, apoi după preț, apoi după cantitate
- **Price Strategy** - un distribuitor își alege producătorii prioritizând doar după preț, apoi după cantitate
- **Quantity Strategy** - un distribuitor își alege producătorii prioritizând după cantitatea de energie oferită per distribuitor

Dacă și tipul și prețul și cantitatea sunt identice atunci producătorii se aleg în ordinea id-urilor lor.

Detalii de implementare

Mecanismul simulării și părțile de input și output sunt similare celor de la prima etapă. Simularea se bazează pe luni (runde), al căror număr este fixat din input (`numberOfTurns`), și se termină când au fost rulate `numberOfTurns + 1` runde și se afișează starea curentă a simulării. În cazul în care toți distribuitorii dau faliment, simularea se va închide.

Input, Output, Rulare

Atât inputul, cât și outputul vor fi de tip `json`. Pentru exemple despre cum să citiți/scrieți fișiere JSON folosind biblioteca jackson aveți acest [tutorial pe wiki](#). Puteți folosi și alte biblioteci pentru citirea lor însă să cereți întâi responsabililor temei (pe forum) adăugarea acestor jar-uri și pe vmchecker.

[Recent changes](#) [Login](#)

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 - Constructor și referințe
▪ Lab 03 - Agregare și moștenire
▪ Lab 04 - Static, Final, Singleton
▪ Lab 05 - Abstracțizare
▪ Lab 06 - Clase interne
▪ Lab 07 - Overriding, Overloading & Visitor pattern
▪ Lab 08 - Colectii
▪ Lab 09 - Design Patterns
▪ Lab 10 - Design Patterns
▪ Lab 11 - Genericitate
▪ Lab 12 - Java features
▪ Lab 13 - Exceptii
Temă
▪ Tema - VideosDB
▪ 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 în 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 Temă
▪ 2019-2020
▪ 2018-2019
▪ 2017-2018
▪ 2016-2017
▪ 2015-2016
▪ 2014-2015
▪ 2013-2014
Table of Contents
▪ Project - Etapa 2 - Sistem energetic
▪ Obiective
▪ Scenariu
▪ Consumatori
▪ Distribuitori
▪ Producători
▪ Strategii de alegere producători
▪ Detaliile de implementare
▪ Input, Output, Rulare
▪ Simularea
▪ Formule
▪ Design
▪ Indicații
▪ Evaluare
▪ Checkstyle
▪ Upload temă
▪ Resurse și linkuri utile

Inputul este același pentru consumatori și are modificări pentru distribuitorii și producători.

Click pentru exemplu input ↗

Output-ul este același ca în etapa 1 pentru distribuitorii și consumatori, iar pentru producători vom avea update-uri lunare despre căți distribuitori au avut.

Click pentru exemplu output ↗

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](#).

Dacă doriti să verificați individual un test, rulați main-ul din clasa Main dând ca argumente fișierul de intrare și fișierul de output.

Click aici pentru a vedea cum să dați argumente main-ului din IDE ↗

Simularea

Simularea va fi aproape identică cu cea din etapa 1 a proiectului. Apariția producătorilor va introduce o nouă funcționalitate, astfel:

1. La începutul primei luni, distribuitorii își vor alege un producător de la care vor cumpăra energie lunar.
2. Distribuitorii vor observa producătorii de la care iau energie, și își vor replica strategia de alegere producători doar în luniile în care apare un update la vreunul din producători.

Runda inițială:

1. Se încarcă datele de intrare despre distribuitori, consumatori și producători.
 - dacă păstrați liste pentru producători să le aveți sortate crescător în funcție de id-ul lor.
2. Distribuitori
 - a. își aleg producătorii
 - b. își calculează costul de producție
 - c. își calculează costul inițial al contractelor
3. Consumatori : - la fel ca la etapa 1
 - a. își aleg distribuitorii și îi plătesc
4. Distribuitorii obțin bani de la consumatorii și își plătesc costurile, actualizând costul infrastructurii

Flow în fiecare lună:

Începutul lunii:

1. se obțin noile valori din test
2. se actualizează valorile pentru distribuitori
3. se actualizează valorile pentru consumatori
4. consumatorii își aleg distribuitorii și îi plătesc
5. distribuitorii își plătesc costurile

În timpul lunii

1. se actualizează valorile citite din test pentru luna respectivă pentru producători

La sfârșitul lunii

1. toți distribuitorii non-bankrupt își actualizează producătorii dacă e cazul și își calculează costul de producție
 - a. ordinea actualizării se face în ordinea crescătoare a id-urilor distribuitorilor
 - b. în momentul în care unui distribuitor îi vine randul să își actualizeze producătorii, întâi se scoate acest distribuitor de la toți producătorii de la care lăua energie și apoi se aplică strategia de alegere.
2. se rețin căți distribuitori a avut fiecare producător

La finalul simulării se scriu în format json în fișierul de output - informații despre consumatori și distribuitori în mod similar primei etape - pentru fiecare producător căți distribuitori a avut în fiecare lună

💡 În momentul în care un distribuitor devine bankrupt, producătorii vor fi informați ca să nu mai îl dea energie (de exemplu dacă păstrați o listă în producător cu ce distribuitori are, îl scoateți din listă).

Formule

Formulele de la prima etapă rămân nemodificate, și se adaugă calculul costului de producție cerut fiecărui distribuitor:

- `cost = sum (cantitate energie de la producator * pret pe Kw de la producator)`
- `productionCost = Math.round(Math.floor(cost / 10));`

Exemplu:

- Un producător A cu energie de tip solar, oferind 1200 KWh lunar fiecărui distribuitor, la prețul 0.1 pe KWh
- Un producător B cu energie de tip nuclear, oferind 2000 KWh lunar fiecărui distribuitor, la prețul 0.3 pe KWh
- Un producător C cu energie de tip nuclear, oferind 1800 KWh lunar fiecărui distribuitor, la prețul 0.2 pe KWh
- Un distribuitor D care are nevoie de 2200 KWh lunar, strategie de tip **Green**.

Distribuitorul D va lua energie de la A și de la C. Costul producției va fi: $(1200 * 0.1 + 1800 * 0.2)/10 = 48$

Design

Design-ul codului vostru ar trebui să fie cât mai decuplat și ușor de extins și urmărit. În afară de folosirea unor design patterns, în evaluarea temei o să luăm în considerare aspecte legate de ce obiecte folosită, ce relații aveți între ele, care este scopul lor, vizibilitatea cămpurilor și metodelor etc (exemplu de astfel de guidelines: [clean code summary](#))

Design patterns-urile pe care le recomandăm pentru aceasta etapă sunt **Strategy** și **Observer**. În mod evident, strategy merge folosit pentru alegerea producătorilor. Legat de observer, în situația din scenariul temei puteți să îl adaptați în mai multe feluri, nu vă impunem un anumit mod, este la latitudinea voastră cum alegeți să îl folosiți.

Câteva posibilități pentru aplicarea Observer:

- fiecare distribuitor trebuie să afle dacă au apărut modificări la producătorii săi. Puteți avea ca Observers distribuitorii și ca Observables consumatorii. În momentul în care un producător primește de la sistemul de simulare update pt energia oferita, atunci își notifică și distribuitorii. Distribuitorii iau în considerare că s-a efectuat o schimbare iar în luna următoare își realeg producătorii
- ca să nu mai fie relație many-to-many între observeri și observabiles, o mitigare ar fi folosirea unei alte clase intermediere drept Subject care doar tine o strucțură de date cu energia oferită de fiecare producător. Când se modifică energia unui producător, se anunță toți distribuitorii și acesta verifică dacă sunt afectați sau nu.
- putem avea ca observatori producătorii și ca observable sistemul de simulare, care lunar primește schimbările legate de energia oferită de producători.

Pentru a avea deja mecanismul de notificare dintre observers și observables, puteți folosi interfața [Observer](#) și clasa [Observable](#) din java.util. Dacă doriti puteți să vă implementați propriile interfețe/clase. Unul dintre motivele pentru care acest API este deprecated din java 9 pentru că este prea simplu, însă acest lucru îl face potrivit pt tema și laborator. Într-o aplicație reală puteți folosi alte apuri-care sunt mult mai complexe și oferă foarte multe tipuri de obiecte și mecanisme (termenul folosit este *reactive programming*).

Indicații

- Separați concepțele de sine stătătoare în clase separate, nu le îmbinăți - clasele ar trebui să aibă un sigur rol
- Adăptați 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-l extindeți greu
- Vă recomandăm să porniți implementarea de la codul scris în prima etapă a proiectului, la care să adăugați noile funcționalități, dar se poate trimite și o implementare scrisă de la zero, care să nu păstreze codul primei părți, dar care să conțină funcționalitatea primei părți
- **Etapa a doua se poate trimite fără să fi trimis prima etapă a proiectului**, însă va fi nevoie să se implementeze și funcționalitățile primei etape pentru a putea primi punctajul total pe teste; în acest caz se va primi punctaj doar pe a doua etapă
- Pentru calculele ce includ procente și numere cu zecimale, veți folosi doar partea întreagă a numărului.

Evaluare

Punctajul constă din:

- 80p implementare - trecerea testelor
- 10p coding style (vezi checkstyle)
- 5p README
 - Puteți porni de la  template-ul dat în schelet, redenumindu-l în README.md. Template-ul este în română însă puteți scrie readme-ul și în engleză
- 5p folosire git pentru versionarea temei



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.

- -7p dacă nu se folosește Observer
- -5p dacă nu se folosește Strategy
- -1 ... -10 nefolosire concepte OOP, greșeli în folosire etc

Dacă vor apărea depunctări specifice temei în momentul evaluării, nementionate pe pagina cu depunctări generale, ele se vor încadra în limitele de maxim 15 pentru design, 5p pentru readme. Dacă tema nu respectă cerințele, sau are zero design OOP atunci se pot face depunctări suplimentare.

Folosirea git pentru versionare va fi verificată din folderul .git pe care trebuie să îl includeți în arhiva temei. Punctajul se va acorda dacă ați făcut minim 3 commit-uri relevante și cu mesaj sugestiv.

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 = 100 și nr_erori = 200 ⇒ nota_finala = 90
- punctaj_total = 100 și nr_erori = 29 ⇒ nota_finala = 100
- 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
- folderul .git

Resurse și linkuri utile

-  Checker, schelet
- [Tutorial Jackson JSON](#)
- [Tutorial IntelliJ pentru import fișiere jar](#)
- [Indicații pentru teme](#)
- [Recomandări coding style & javadoc](#)

poo-ca-cd/teme/proiect/etapa2.txt · Last modified: 2021/01/17 14:27 by florian.mihalache

 Old revisions  Media Manager  Back to top

 CHIMERIC DE W3C CSS DOKUMIKI GET FIREFOX RSS XML FEED W3C XHTML 1.0