Tema 1 - Sheriff of Nottingham

- Responsabili: Andrei Ene, Veronica Radu,Alexandru Rotaru
- Deadline: ~~06.11.2018~~ 07.11.2018 23:59
- Deadline hard: 13.11.2018 23:59
- Data publicării: 21.10.2018
- Data ultimei modificări: 25.10.2018

Obiective

- familiarizarea cu Java și conceptele de bază ale POO
- fundamentarea practică a constructorilor și a agregării/moștenirii
- folosirea conceptelor de supraîncărcare și suprascriere
- dezvoltarea unor abilități de bază de organizare și design orientat-obiect
- respectarea unui stil de codare si de comentare
- respectarea principiilor SOLID

Cerințe

Se dorește implementarea unei versiuni minimaliste a jocului Sheriff of Nottingham, utilizând conceptele POO.

Descrierea jocului

Goana după aur a prințului John a mers prea departe! Este imposibil pentru un comerciant să poată să își mai câștige existența, taxele fiind la fel de mari ca pentru un cumpărător obișnuit. Acum, prințului i s-a alăturat și avarul șerif din Nottingham, care verifică pe oricine îi calcă teritoriul, adică verifică

toate bunurile ilegale ale unui comerciant și le păstrează după bunul său plac.

Este bine că tu îl cunoști mai bine pe acest șerif decât prințul și știi că acea persoană, aparent intimidantă când stă în fața porții orașului, nu refuză o mită generoasă atunci când i se oferă. Tot ce ai nevoie este o strategie bine gandită, iar rolul tău acum este să o găsesti pe cea mai bună.
Pentru acest lucru, se va simula un joc cu trei jucători, fiecare jucător având un anumit tip de strategie. Pentru a obține un rezultat cât mai obiectiv, fiecare jucător va avea două roluri: **șerif** și **comerciant**. Fiecare jucător urmăreste să își maximizeze profitul, astfel încât la finalul jocului el să fie declarat jucătorul cu cea mai bună strategie.

Atunci când este în rolul de comerciant, el incearcă să aducă pe piața din Nottingham cât mai multe bunuri (legale și ilegale) astfel încât să își mărească câștigul. În rolul de șerif, el are posibilitatea de controla sau nu sacul cu bunurile unui comerciant. El poate fi mituit de acești sau poate controla oricâți comercianți (toți sau niciunul), el la rândul lui urmărind propriul câștig.

Implementare

Jocul este organizat în **runde**. La fiecare runda unul din **jucători** va avea rolul de șerif, iar ceilalți vor avea rolul de comercianți. Jocul se încheie atunci când fiecare jucător a avut rolul de șerif **de două ori**. Fiecare jucător va avea în visteria sa la început **50 monede** ce pot fi folosite pentru a mitui un șerif. Numarul maxim de jucatori este de **3**.

Runda de joc

Fiecare rundă este împărțită în patru etape: crearea sacului, declararare bunurilor, inspecția și recompletarea bunurilor din mână.

Crearea sacului

La fiecare rundă, fiecare jucător va deține **6 cărți** (bunuri). El va trebui să selecteze între **1 și 5 cărți** (de orice tip) din cele 6 și să le pună într-un sac pentru a le înmâna șerifului.
De asemenea, pentru a fi mai convingător, în funcție de strategia adoptata, el poate adăuga și o **sumă de bani** (mită) pentru a convinge șeriful să nu îi controleze sacul.

Declararea bunurilor

La această etapă, fiecare jucător predă sacul șerifului și își declară bunurile, indicând tipul lor. Indiferent de bunurile din sac, el trebuie să declare **un singur tip** și poate să spună sau nu adevărul.

Inspecția

Atunci când șeriful primește sacii de la comercianți, el trebuie sa aleagă pe care îi va controla (poate să controleze oricâți comercianți), respectând însă următoarele reguli:

- dacă a prins un mincinos, atunci el câștigă bani (***penalty***) pentru fiecare bun ilegal sau nedeclarat. Fiecare bun ilegal sau nedeclarat va fi confiscat. Un bun confiscat va fi adăugat mereu la finalul grămezii cu celelalte cărți rămase în joc.
- dacă a controlat un comerciant sincer, atunci el trebuie sa plătească o sumă proporțională cu numărul de bunuri declarate de acesta (***assetsCount * penalty***)
- dacă alege să accepte mita, atunci șeriful va adăuga acea sumă de bani la câștigul propriu, iar comerciantul va trece necontrolat, deci își va putea adăuga toate bunurile din sac pe tarabă.
- **daca se controleaza sacul implicit se refuza mita daca exista(aceasta intorcandu-se la comerciant)**
- **daca se nu se controleaza sacul, implicit se accepta mita (daca exista)**

Recompletarea bunurilor din mana

La finalul rundei fiecare jucător își pune bunurile aduse pe taraba și își completează cărțile din mână pana ajunge sa dețină din nou 6 cărți (bunuri).

Descrierea bunurilor

Un comerciant are acces la două tipuri de bunuri (reprezentate sub forma unor carți de joc):

- bunuri legale: ***Apple, Cheese, Bread, Chicken***
- bunuri de contrabandă (ilegale): ***Silk, Pepper, Barrel.***

Fiecare bun are urmatoarele caracterstici:

- ***profit*** - suma de bani pe care comerciantul o va câștiga la finalul jocului dacă aduce în Nottingham bunul respectiv;
- ***penalty*** - suma de bani pe care trebuie să o plătească un comerciant atunci când este inspectat de șerif și prins cu bunuri nedeclarate sau suma de bani pe care trebuie sa o plătească șeriful comerciantului pe care l-a inspectat pe nedrept.

De asemenea la finalul jocului se calculează pentru fiecare tip de bun legal un clasament în funcție de numărul de bunuri deținute de acel tip. Jucătorul cu cele mai multe bunuri este declarat **rege**, iar următorul este declarat **regină**. Mai jos aveți o listă cu punctajele acordate în funcție de tipul bunului și locul în clasament. Dacă un jucator nu reușește să intre în primele 2 locuri din clasament nu primește bonusuri.

Type of Good	King’s Bonus	Queen’s Bonus
Apple	20	10
Cheese	15	10
Bread	15	10
Chicken	10	5

Descrierea strategiilor

Fiecare jucător va avea o strategie proprie de joc (3 ca șerif, cât și pentru rolul de comerciant).

Jucătorul de bază (Base Strategy)

În rolul de **comerciant**, el este jucătorul onest, corect, care *spune mereu adevărul*. În funcție de cărțile pe care le are în mână, el va cauta tipul de cărți **cel mai frecvent**. Dacă vor exista mai multe tipuri de cărți cu aceeași frecvență, va selecta tipul de carte care i-ar aduce un profit mai mare. In cazul in care sunt mai multe bunuri cu aceeasi frecventa si acelasi profit se alege primul bun din mana. Dacă are doar cărți ilegale, el va aduce în sacul său o singura carte, încercând să își minimizeze fapta ilegala (va adăuga totuși cartea care îi aduce cel mai mare profit și va declara că în sac se află **mere**)
Ca **șerif**, el *va controla toți ceilalți jucători*, la fiecare rundă, nu va accepta bani de la ceilalți comercianți și deși va risca să rămână fără bani, el va încerca sa împiedice toate bunurile ilegale care ar putea fi aduse în Nottingham.

Jucatorul lacom (Greedy Strategy)

Este un **jucător de bază** însă care atunci când este **șerif** este ușor de mituit. El inspectează toți comercianții, mai puțin pe cei care îi oferă mită.
Când este în rolul de **comerciant** el *nu oferă mită*, deși **în rundele pare**, după ce aplică **strategia de bază**, el adaugă în sacul său (dacă nu are deja 5 bunuri în sac) un **bun ilegal**, alegând bineînțeles cartea ilegală cu profitul cel mai mare. Dacă în runda respectivă el nu are nicio astfel de carte, el va juca doar strategia de bază. Prima rundă se consideră impară.

Jucătorul care mituiește (Bribe Strategy)

Jucătorul care mituiește este cel care încearcă să îi păcălească pe ceilalți, dar care la un moment dat riscă să se păcălească pe sine. În rolul de **comerciant**, el *va da mită* cât timp venitul îi va permite și va încerca mereu sa pună cat mai multe cărți ilegale (maxim 5, cele ce profitul cel mai mare), după următoarele reguli:

- va da **5 monede** atunci când în sac va adăuga **una sau două** bunuri ilegale;
- va da **10 monede** atunci când în sac va adăuga **mai mult de două** cărți ilegale;
- va declara bunurile ca fiind **mere**;

Dacă va rămâne fără bani, el nu va putea da mită, deci el va juca corect (va juca la fel ca jucătorul cu **strategia de bază**). De asemenea, dacă nu va avea niciun bun ilegal, el va aplica tot strategia de bază.
Ca **șerif**, el va verifica mereu comerciantul din **stânga**, respectiv **dreapta** sa. Astfel, pentru un joc în care ordinea stabilita este *basic*, *greedy*, *bribe*, dacă șeriful de la runda curentă este *bribe*, acesta va verifica jucătorul cu strategia *greedy* (din stanga) și pe cel cu strategia *basic* (din dreapta).

Precizări

Descrierea inputului

Fișierul de input va conține următoarele elemente:

- Ordinea jucătorilor: un vector de string-uri cu numele celor 3 strategii. Jucătorul cu strategia ce corespunde primului element din vector va fi jucătorul care va fi primul în rolul de șerif. Exemplu: [“basic”, “bribe”, “greedy”]
- Cărțile din joc: un vector cu 216 elemente. Un element reprezinta un id al unei cărți.

Mai jos avem un tabel in care sunt prezentate informatiile despre bunuri.

Id	Asset	Type	Profit	Penalty	Bonus
0	Apple	Legal	2	2	Nothing
1	Cheese	Legal	3	2	Nothing
2	Bread	Legal	4	2	Nothing
3	Chicken	Legal	4	2	Nothing
10	Silk	Illegal	9	4	3 * Cheese
11	Pepper	Illegal	8	4	2 * Chicken
12	Barrel	Illegal	7	4	2 * Bread

Cărțile ilegale sunt cele valoroase. Pe lângă profitul mai mare, unele dintre ele aduc și un bonus deținătorului, iar acel bonus va fi de asemenea adăugat la punctajul final.

Exemplu: Pentru un jucător care deține o carte de tip **Pepper**, la punctajul final se va adăuga profitul corespunzător (**8 monede**), cat și bonusul (**2 cărți de tip Chicken**). Așadar, o carte de tip Pepper va aduce la scor o creștere de 8 + 2 * 4 = 16 puncte.

Descrierea outputului

La final jocului ne intereseaza un simplu clasament in care avem numele jucatorului impreuna cu punctajul sau final. Acest clasament trebuie afisat sortat dupa numarul de puncte descrescator.

```
BASIC: 140
BRIBED: 91
GREEDY: 90
```

Clasamentul se va afisa pe consola.

Calcul scor final

La final, fiecare jucător își va verifica bunurile aduse în Nottingham și pentru fiecare bun ilegal va adăuga (dacă este cazul) pe taraba sa bonusul adus de respectiva carte.

Exemplu: Un jucător care deține o carte de tip **Silk**, va adăuga pe taraba sa **3** bunuri de tip **Cheese**.

Scorul final al unui jucător va fi reprezentat de suma dintre:

- numărul de monede rămase în **visterie**;
- profitul provenit de la **bunurile legale** aduse în Nottingham;
- profitul provenit de la **bunurile ilegale** aduse în Nottingham;
- **bonusul** în funcție de rolul (rege/regină) pe care jucătorul poate să îl dețină pentru un anumit tip de bun.

Exemplu

```
[12, 10, 12, 11, 2, 2, 10, 0, 1, 1, 1, 3, 2, 0, 2, 11, 2, 10, 3, 0, 2, 1, 0,
11, 3, 10, 0, 0, 12, 1, 0, 2, 1, 12, 0, 1, 11, 0, 12, 2, 11, 2, 1, 2, 1, 10,
0, 11, 11, 2, 0, 0, 0, 0, 10, 3, 10, 1, 0, 0, 0, 12, 0, 1, 0, 3, 1, 2, 2, 2,
2, 0, 2, 11, 2, 2, 10, 12, 3, 1, 1, 1, 3, 1, 2, 1, 2, 2, 1, 2, 0, 1, 11, 3,
3, 0, 2, 1, 2, 0, 3, 3, 3, 12, 2, 12, 2, 0, 11, 0, 2, 11, 2, 1, 3, 1, 3, 2,
12, 2, 12, 3, 2, 2, 3, 10, 0, 11, 0, 3, 2, 11, 2, 11, 3, 0, 2, 0, 0, 0, 3,
10, 0, 0, 0, 11, 12, 12, 1, 12, 3, 1, 3, 10, 10, 1, 2, 12, 0, 3, 1, 0, 2, 0,
1, 1, 3, 0, 3, 12, 2, 0, 10, 1, 10, 10, 12, 0, 1, 12, 2, 10, 1, 10, 0, 0,
11, 0, 1, 1, 1, 3, 0, 0, 3, 12, 11, 2, 1, 0, 1, 12, 11, 1, 1, 11, 1, 0, 0,
10, 10, 0, 1, 0, 10, 11]
["bribed", "basic", "greedy"]
```

Făcând conversia intre id-uri și tipurile de cărți de joc, vectorul va fi transformat în:

```
  [Barrel, Silk, Barrel, Pepper, Bread, Bread, Silk, Apple, Cheese, Cheese,
Cheese, Chicken, Bread, Apple, Bread, Pepper, Bread, Silk, Chicken, Apple,
Bread, Cheese, Apple, Pepper, Chicken, Silk, Apple, Apple, Barrel, Cheese,
Apple, Bread, Cheese, Barrel, Apple, Cheese, Pepper, Apple, Barrel, Bread,
Pepper, Bread, Cheese, Bread, Cheese, …. ]
```

Un exemplu de desfășurare a rundelor: exemplu_sheriff_of_nottingham

Tutorial colecții

Pentru realizarea implementării veți avea nevoie de anumite structuri de date pentru a stoca elementele sau a le ordona. Pachetul **java.util** oferă mai multe clase și interfețe pentru reprezentarea şi manipularea colecţiilor. În continuare sunt prezentate două exemple ce ilustrează folosirea interfețelor **Map** și **List**.

Cookie.java

```
class Cookie {
    String type;
    int weight;

    Cookie(String type, int weight) {
        this.type = type;
        this.weight = weight;
    }
}
```

Pentru a crea o listă cu obiecte de tip Cookie procedam astfel:

Test.java

```
class Test {
    public static void main(String[] args) {
        List<Cookie> cookies = new LinkedList<Cookie>();
        cookies.add(new Cookie("chocolate", 125));
        cookies.add(new Cookie("peanuts", 100));
        cookies.add(new Cookie("vanilla", 120));

        for(Cookie cookie : cookies) {
            System.out.println(cookie.type);
        }
    }
}
```

Dacă vrem să avem obiectele sortate în funcție de câmpul **weight** avem nevoie de un comparator pentru obiectele de tip Cookie:

CookieComparator.java

```
class CookieComparator implements Comparator<Cookie> {
    @Override
    public int compare(Cookie c1, Cookie c2) {
        return c1.weight - c2.weight;
    }
}
```

Test.java

```
class Test {
    public static void main(String[] args) {
        CookieComparator cookieComparator = new CookieComparator();
        List<Cookie> cookies = new LinkedList<Cookie>();
        cookies.add(new Cookie("chocolate", 125));
        cookies.add(new Cookie("peanuts", 100));
        cookies.add(new Cookie("vanilla", 120));

        Collections.sort(cookies, cookieComparator);

        for(Cookie cookie : cookies) {
            System.out.println(cookie.type);
        }
    }
}
```

Atunci când avem nevoie să reținem asocieri de tip **cheie - valoare** folosit interfața Map. Următorul exemplu numără aparițiile fiecărui cuvânt dintr-un vector de String-uri:

```
String[]  flavors = {"chocolate", "chocolate", "vanilla", "peanuts",
```

```
"strawberry", "peanuts"};
Map<String, Integer> countFlavors = new HashMap<String, Integer>();

for (String flavour : flavors) {
    countFlavors.put(flavour, countFlavors.getOrDefault(flavour, ) + 1);
}

for (String key : countFlavors.keySet()) {
    System.out.println(key + ": " + countFlavors.get(key));
}
```

Bonus

Se acordă **20 puncte** bonus pentru implementarea unei strategii care să obțină mai multe puncte decât cele patru strategii descrise mai sus.

In cadrul bonusului jocul va avea pana la **4** jucatori, al patrulea se va numi **wizard**. Astfel inputul devine:

```
[12, 10, 12, 11, 2, 2, 10, 0, 1, 1, 1, 3, 2, 0, 2, 11, 2, 10, 3, 0, 2, 1, 0,
11, 3, 10, 0, 0, 12, 1, 0, 2, 1, 12, 0, 1, 11, 0, 12, 2, 11, 2, 1, 2, 1, 10,
0, 11, 11, 2, 0, 0, 0, 0, 10, 3, 10, 1, 0, 0, 0, 12, 0, 1, 0, 3, 1, 2, 2, 2,
2, 0, 2, 11, 2, 2, 10, 12, 3, 1, 1, 1, 3, 1, 2, 1, 2, 2, 1, 2, 0, 1, 11, 3,
3, 0, 2, 1, 2, 0, 3, 3, 3, 12, 2, 12, 2, 0, 11, 0, 2, 11, 2, 1, 3, 1, 3, 2,
12, 2, 12, 3, 2, 2, 3, 10, 0, 11, 0, 3, 2, 11, 2, 11, 3, 0, 2, 0, 0, 0, 3,
10, 0, 0, 0, 11, 12, 12, 1, 12, 3, 1, 3, 10, 10, 1, 2, 12, 0, 3, 1, 0, 2, 0,
1, 1, 3, 0, 3, 12, 2, 0, 10, 1, 10, 10, 12, 0, 1, 12, 2, 10, 1, 10, 0, 0,
11, 0, 1, 1, 1, 3, 0, 0, 3, 12, 11, 2, 1, 0, 1, 12, 11, 1, 1, 11, 1, 0, 0,
10, 10, 0, 1, 0, 10, 11]
["bribed", "basic", "wizard", "greedy"]
```

Corectarea bonusului

Testele de input de la bonus for fi doar cu 2 jucatori: unul de la bonus (“wizard”) si altul la alegere din cei 3 (“x”)*. Si pentru acelasi pachet de carti avem doua teste: primul cu ordinea:[“wizard”, “x”] si al doilea cu ordinea: [“x”, “wizard”].
Vom calcula pentru primul test: test1Rez = scorWizard - scorX si pentru al doilea test la fel: test2Rez = scorWizard - scorX.
Acordam bonusul daca: test1Rez + test2Rez > 0. Ceea ce inseamna ca pentru acelasi set de carti “wizard” s-a descurcat mai bine fata de jucatorul “x”.

- “x” va fi unul dintre: “basic”, “greedy”, “bribed”

Va puteti folosit de strategia Royal:

Jucătorul care urmărește bonusurile (Royal Strategy)

Așa cum am menționat anterior, la finalul jocului se vor acorda bonusuri semnificative jucătorilor care dețin cele mai multe cărți de același tip. Un jucător cu Royal Strategy încearcă permanent să monitorizeze bunurile sale, precum și pe cele deținute de ceilalți jucători.
Atunci când este **comerciant**, pentru a lua o decizie, va aplica următorul raționament:

- Va alege tipul de carte cu scorul cel mai mare. Scorul unui bun pentru jucătorul curent va fi calculat după următoarea functie de scor:

```
score = (assetCountOnMerchantStand + assetCountInHand) * assetProfit +
kingBonus + queenBonus
```

assetCountOnMerchantStand - numărul de bunuri din tipul respectiv pe care jucătorul le are pe tarabă;
assetCountInHand - numărul de cărți din tipul respectiv pe care jucătorul le are în mână;
assetProfit - profitul corespunzătorul tipului de bun pentru care se calculează scorul;
kingBonus - bonusul ce se va obține la finalul jocului dacă jucătorul curent va ajunge rege pentru tipul de carte pentru care se calculează scorul;
queenBonus - bonusul ce se va obține la finalul jocului dacă jucătorul curent va ajunge regină pentru tipul de carte pentru care se calculează scorul;

In cazul in care nu exista bunuri legale la dispozitie va selecta cel mai valoaros bun ilegal.Daca avem exista mai multe astefel de bunuri se transmite doar unul dintre ele.

În rolul de **șerif**, un astfel de jucător consideră că sunt suspecți sacii care conțin cel puțin **patru** bunuri și de aceea îi va verifica mereu.

Punctaj

- 60p trecerea testelor
- 20p **coding style** (vezi checkstyle)
- 15p design și organizare
- 5p README clar, concis, explicații axate pe design (flow, interacțiuni)
- 20p bonus

Checkstyle

Mai jos aveți câteva exemple de concepte de avut în vedere pentru a trece testul de coding-style/checkstyle

- numele fișierelor (ref)
- organizarea fișierelor (ref)
- indentarea pe verticală și orizontală (ref)
- declarațiile și inițializările (ref)
- numirea claselor, variabilelor, metodelor, etc. (ref)
- tratarea cazurilor speciale (ref)
- respectarea unui coding style (nu neapărat acesta, important este să fiți **consistenți** și **consecvenți**)

Pentru a fi eligibil de bonus, tema trebuie să treacă testul de coding-style executat de Checkstyle însă dacă este picat și numărul de erori depistate depășește 30 (o treime din punctajul maxim, fără bonus), 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 = 80`
- `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 = 60`

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 !!

Structura arhivei

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

- fișierul README
- fișierele din scheletul temei
- alte fișiere **organizate** cu implementarea **voastră** (doar fișierele sursă **.java**)

Resurse

Pe git gasiti un set de teste și un schelet de cod care face parsarea din fișierul de intrare.
https://github.com/oop-pub/teme/tree/master/tema1/

Actualizari

- 26.10.2018 actualizarea testelor datarita faptului ca anumite referinte nu aveau aceasi versiune ca testele oferite.

Referințe

- Sheriff of Nottingham Rules
- SOLID Principles in Java
- Colectii
- Java ArrayList of Object Sort Example

- Tutorial checkstyle