



You are here: [Programare Orientată pe Obiecte](#) » [Teme](#) » **[Tema - Sheriff of Nottingham](#)**

Administrativ

- [Regulament](#)
- [Echipa](#)
- [Indicații pentru teme](#)
- [Indicații pentru testul practic](#)
- [Recomandări cod](#)

Laboratoare

- [Lab 01 - Java Basics](#)
- [Lab 02 - Constructori și referințe](#)
- [Lab 03 - Agregare și moștenire](#)
- [Lab 04 - Static, Final, Singleton](#)
- **[Lab 05 - Clase abstracte și interfețe](#)**
- [Lab 06 - Clase interne](#)
- [Lab 07 - Overriding, Overloading & Visitor pattern](#)
- [Lab 08 - Colecții](#)
- [Lab 09 - Genericitate](#)
- [Lab 10 - Excepții](#)
- [Lab 11 - Design Patterns](#)
- [Lab 12 - Design Patterns](#)
- [Recapitulare](#)

Tema

- [Tema - Sheriff of Nottingham](#)
- [Etapa 1 - League of OOP](#)
- [Etapa 2 - League of OOP](#)

Teste

- [Teste grilă](#)
- [Teste practice](#)

Resurse utile

- [Instalare IntelliJ Idea](#)
- [Activare IntelliJ Idea](#)
- [Instalare Eclipse](#)
- [Demo proiect IntelliJ Idea](#)
- [Demo proiect Eclipse](#)
- [Tutorial checkstyle](#)

Alte resurse

- [Exerciții](#)
- [POO și Java](#)
- [JUnit](#)
- [Organizarea surselor și controlul accesului](#)
- [Tutorial I/O](#)
- [JSON & Jackson](#)
- [Double Dispatch - scurt tutorial](#)
- [Reflection](#)

Arhiva Teme

- [2018-2019](#)

- 2017-2018
- 2016-2017
- 2015-2016
- 2014-2015
- 2013-2014

teme:tema

Table of Contents

- ♦ Tema - Sheriff of Nottingham
 - ♦ Obiective
 - ♦ Cerințe
 - ♦ Descrierea jocului
 - ♦ Implementare
 - ♦ Runda de joc
 - ♦ Descrierea bunurilor
 - ♦ Descrierea strategiilor
 - ♦ Precizări
 - ♦ Descrierea inputului
 - ♦ Descrierea outputului
 - ♦ Tutorial colecții
 - ♦ Punctaj
 - ♦ Checkstyle
 - ♦ Structura arhivei
 - ♦ Resurse
 - ♦ Referințe

Tema - Sheriff of Nottingham

- Responsabili: ✉ [Gabriel Țuculină](#), ✉ [Florin Mihalache](#)
- Deadline: 10.11.2019 - 23:55
- Deadline hard: 17.11.2019 - 23:55
- Data publicării: 19.10.2019
- Data ultimei modificări: 19.10.2019

Obiective

- familiarizarea cu Java și conceptele de bază ale POO
- utilizarea 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 și de comentare
- respectarea principiilor 🌐 [SOLID](#)

Cerințe

Se dorește implementarea unei versiuni minimaliste a jocului Sheriff of Nottingham, utilizând conceptele POO. Regulile jocului nu sunt fix cele din boardgame, ele sunt adaptate pentru tema.



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 gândită, iar rolul tău acum este să o găsești pe cea mai bună.

Pentru acest lucru, se va simula un joc cu **maxim 7 jucători**, așezați în cerc, 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 urmărește 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, jucătorul încearcă 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 aceștia sau poate controla comerțanții (toți sau niciunul), el la rândul lui urmărind propriul câștig.

Implementare

Jocul este organizat în **runde**. La fiecare sub-rundă (**o rundă având n sub-runde, n fiind numărul de jucători**), doar unul dintre jucători va avea rolul de **șerif**, iar toți ceilalți vor avea rolul de **comercianți**, fiecare comerciant extragând câte 10 carti odata. Jocul se încheie atunci când fiecare jucător a avut rolul de șerif de **maxim 5 ori**. Fiecare jucător va avea în visteria sa la început **80 de monede** ce pot fi folosite pentru a mitui un șerif. Numărul maxim de jucatori este de 7.

Runda de joc

Fiecare sub-rundă este împărțită în patru etape: crearea sacului, declararea bunurilor, inspecția și reumplerea bunurilor din mână.

Crearea sacului

La fiecare sub-rundă, fiecare jucător va deține **10 cărți** (bunuri). El va trebui să selecteze între **0 și 8 cărți** (fiecare carte poate să fie de orice tip) din cele 10 și să le pună într-un sac pentru a le înmâna șerifului. De

asemenea, pentru a fi mai convingător, în funcție de strategia adoptată, 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. Bunurile care nu sunt alese, se pot considera **arse**.

Inspecția

Atunci când șeriful primește sacii de la comercianți, el trebuie să aleagă pe care dintre comercianți îi va controla (poate să oricâți comercianți, inclusiv pe toți sau pe niciunul), 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 (returnata de metoda `getAssetIds()` din clasa `GameInput`) cu celelalte cărți rămase în joc.
- dacă a controlat un comerciant sincer, atunci el trebuie să 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.
- dacă se controlează sacul implicit se refuză mita dacă există (aceasta întorcându-se la comerciant)

Recompletarea bunurilor din mână

La finalul sub-rundei fiecare jucător își pune bunurile aduse pe taraba și își completează mâna cu următoarele 10 cărți (bunuri).

Descrierea bunurilor

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

- bunuri legale: ***Apple, Cheese, Bread, Chicken, Tomato, Corn, Potato, Wine, Salt, Sugar***
- bunuri de contrabandă (ilegale): ***Silk, Pepper, Barrel, Beer, Seafood***

Fiecare bun are următoarele caracteristici:

- **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 să 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 jucător 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	19	9
Bread	18	9
Chicken	17	8
Tomato	16	7

Corn	15	6
Potato	14	5
Wine	13	4
Salt	12	3
Sugar	11	2

Descrierea strategiilor

Fiecare jucător va avea o strategie proprie de joc (una din cele descrise mai jos – atât ca șerif cât și ca și 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. În cazul în care sunt mai multe bunuri cu aceeași frecvență și același profit se alege bunul cu id-ul cel mai mare. Dacă are doar cărți ilegale, el va aduce în sacul său o singură carte, încercând să își minimizeze fapta ilegală (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 sub-rundă, nu va accepta bani de la ceilalți comercianți și deși va risca să rămână fără bani, el va încerca să împiedice toate bunurile ilegale care ar putea fi aduse în Nottingham. Dacă a rămas fără bani (are o sumă mai mică de 16), acesta nu va mai controla niciun comerciant până la finalul turului său ca șerif, dar nu va accepta nici mita acestora. Practic își va termina turul ca șerif.

Jucătorul 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 8 bunuri în sac) un **bun ilegal**, alegând bineînțeles cartea ilegală cu profitul cel mai mare. Dacă într-o sub-rundă **a rundei pare** el nu are nicio astfel de carte, el va juca doar strategia de bază în sub-runda respectiva. 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 să pună cât mai multe cărți ilegale (maxim 8, cele ce au profitul cel mai mare) și eventual să completeze cu legale, 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 completa cu cartile legale cele mai valoroase, iar la egalitate, cu cele cu id-ul cel mai mare, în limita permisă de banii pe care îi are
- va declara bunurile ca fiind **mere**;

Dacă ar 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. El își va completa sacul (ce îi da șerifului la verificare) în funcție de penalty-ul care s-ar acumula. Totuși, el nu va lua un bun dacă i-ar aduce penalty-ul acumulat suma de bani pe 0. Ca **șerif**, el va verifica mereu doar comerciantul din **stânga**, respectiv **dreapta** sa, în aceasta ordine. De la restul comercianților va încerca, **ulterior**, să ia mita dacă aceștia i-ar oferi-o, dar altfel **nu** îi verifica.

Precizări

Descrierea inputului

Fișierul de input va conține următoarele elemente:

- numărul rundelor de joc
- numărul de jucători
- ordinea jucătorilor: un vector de string-uri de dimensiune maxima 7, 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"]
- numărul de cărți
- cărțile din joc: un vector cu elemente ce reprezinta id-uri al bunurilor.

Exemplu de input:

```
1
2
bribed bribed
20
5 3 5 23 6 9 0 3 0 9 24 8 9 21 8 5 7 1 0 0
```

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
4	Tomato	Legal	3	2	Nothing
5	Corn	Legal	2	2	Nothing
6	Potato	Legal	3	2	Nothing
7	Wine	Legal	5	2	Nothing
8	Salt	Legal	2	2	Nothing
9	Sugar	Legal	3	2	Nothing
20	Silk	Illegal	9	4	3 * Cheese
21	Pepper	Illegal	8	4	2 * Chicken
22	Barrel	Illegal	7	4	2 * Bread
23	Beer	Illegal	6	4	4 * Wine
24	Seafood	Illegal	12	4	2 * Tomato + 3 * Potato + 1 * Chicken

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 interesează un simplu clasament în care avem id-ul jucătorului (al câtelea a fost în vectorul inițial), strategia utilizată de acesta, împreună cu punctajul său final. Acest clasament trebuie afișat sortat după numărul de puncte descrescător.

```
2 BASIC 140
4 BRIBED 91
1 BASIC 89
0 GREEDY 70
3 GREEDY 20
```

Clasamentul se va afișa pe consolă.

Calcul scor 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.

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.

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 și manipularea colecțiilor. 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"};
Map<String, Integer> countFlavors = new HashMap<String, Integer>();

for (String flavour : flavors) {
    countFlavors.put(flavour, countFlavors.getOrDefault(flavour, 0) + 1);
}

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


```

Punctaj

- 60p trecerea testelor
- 20p **coding style** (vezi [checkstyle](#))

- 15p design și organizare
- 5p README clar, concis, explicații axate pe design (flow, interacțiuni)

Checkstyle

Unul din obiectivele temei este învățarea respectării code-style-ului limbajului pe care îl folosiți. Aceste convenții (de exp. cum numiți fișierele, clasele, variabilele, cum indentați etc) 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 = 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 și **repetarea materiei** atât pentru sursă(e) cât și pentru destinație(ii), fără excepție.

Structura arhivei

Arhiva pe care o veți urca pe  [VMChecker](#) va trebui să conțină în directorul rădăcină:



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

Resurse

Pe GitHub găsiți un schelet de cod care face parsarea din fișierul de intrare.  <https://github.com/oop-pub/teme/tree/skel-tema-2019/tema/schel>

Pentru a folosi fileio din schelet, urmăriți tutorialul de aici.  <http://elf.cs.pub.ro/poo/laboratoare/tutorial-io>

Exemple de rezolvari pe [foaie](#) a testelor de input:

- 1round2players-mixed (basic-basic):  <https://prntscr.com/pmpjwg>
- 2rounds2players-legal (basic-greedy):  <https://prntscr.com/pmpp71>

Referințe

-  [Sheriff of Nottingham Rules](#)
-  [SOLID Principles in Java](#)
-  [Colectii](#)
-  [Java ArrayList of Object Sort Example](#)
- [Tutorial checkstyle](#)

