



You are here: [Programare Orientată pe Obiecte](#) » [Teme](#) » [proiect](#) » **[Proiect - Etapa 1 - League of OOP](#)**

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:proiect:etapa1

Table of Contents

- ♦ [Proiect - Etapa 1 - League of OOP](#)
 - ♦ [Obiective](#)
- ♦ [Scenariu](#)
 - ♦ [Hartă](#)
 - ♦ [Eroi](#)
 - ♦ [Experiență și level up](#)
 - ♦ [Abilități](#)
 - ♦ [Pyromancer](#)
 - ♦ [Knight](#)
 - ♦ [Wizard](#)
 - ♦ [Rogue](#)
 - ♦ [Mecanism de joc](#)
 - ♦ [Mențiuni](#)
 - ♦ [Aproximări și rotunjiri](#)
- ♦ [Cerințe](#)
 - ♦ [Double Dispatch](#)
 - ♦ [Input](#)
 - ♦ [Output](#)
 - ♦ [Exemplu input/output](#)
 - ♦ [API](#)
 - ♦ [Indicații](#)
 - ♦ [Git](#)
 - ♦ [Punctaj](#)
 - ♦ [Checkstyle](#)
 - ♦ [Structura arhivei](#)
 - ♦ [Link-uri utile](#)

Proiect - Etapa 1 - League of OOP

- **Data publicarii:** 10.11.2019 21:00
- **Data ultimei modificari:** 10.11.2019 21:00
- **Deadline soft:** 01.12.2019 23:55
- **Deadline hard:** 08.12.2019 23:55
- **Responsabili:** ✉ [Ionuț Bîrsu](#), ✉ [Laurențiu Stamate](#), ✉ [Ana-Maria Micu](#), ✉ [Bogdan-Cristian Firuți](#)

Obiective

- familiarizarea cu Java și conceptele de bază ale POO
- fundamentarea practică a constructorilor și a agregării/moștenirii
- 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
- respectarea unui stil de codare și de comentare

Scenariu

Suntem într-un joc MMO-style. Eroi noștri își petrec viața într-un univers 2D, pe care îl explorează și în care se dezvoltă.

După cum este cunoscut din jocurile MMO și RPG, personajele noastre sunt diverse, fiecare aparținând unei anumite clase și având un anumit set de abilități la care se pricepe cel mai bine și pe care le poate dezvolta cel mai repede; le vom analiza în detaliu mai jos.

La începutul jocului, eroii noștri sunt plasați pe hartă în locuri bine definite. Desfășurarea jocului presupune existența unor runde cu durată unitate în care toți eroii vor executa câte o mișcare bine definită pe hartă.

Când doi eroi ajung în același loc, ei se vor lupta. În cadrul unei runde, fiecare erou combatant își va folosi toate abilitățile disponibile împotriva adversarului, o singură dată. După luptă, ei își vor vedea de drum începând cu runda următoare.

La sfârșitul jocului (un număr stabilit de runde, cu locații inițiale și mișcări stabilite), programul nostru se va uita la eroii rămași în viață.

Hartă

Harta pe care se desfășoară jocul este un careu 2D (dreptunghi), compus din locații ("pătrățele") de dimensiune unitate. Fiecare locație are un anumit tip, ceea ce îi conferă un set de proprietăți; tipurile de locații disponibile sunt:

- Land
- Volcanic
- Desert
- Woods

Eroi

Există mai multe tipuri de eroi în LOOP:

- Knight
- Pyromancer
- Rogue
- Wizard

Toți eroii au două proprietăți de bază și anume un număr de hit points(HP, "viață") și un număr de puncte de experiență (XP). Pe lângă acestea, există și un mecanism de level-up în funcție de experiența câștigată în urmă luptelor.

Experiență și level up

Toate personajele au **XP inițial = 0**, corespunzător nivelului 0. În momentul în care un personaj câștigă o luptă (își omoară adversarul), XP-ul său va crește după următoarea formulă:

```
XP_Winner = XP_Winner + max(0, 200 - (Level_winner - Level_loser) * 40)
```

După cum se poate observa și din formulă, câștigarea unei lupte cu un adversar care este cu cel puțin 5 nivele mai mic nu va aduce nici un plus de XP, dar câștigul în fața unui oponent de nivel mai mare poate aduce chiar mai mult de 200XP.

Observație: Se poate întâmpla ca eroii să se omoare reciproc. În acest caz, ambii vor primi XP-ul corespunzător.

Creșterea în nivel are loc în momentul în care se depășește un prag de XP, ce se va calcula după următoarea formulă:

$$XP_level_up = 250 + Level_curent * 50$$

Spre exemplu, dacă în urma primei runde un personaj câștigă 360 XP, acesta va avansa direct la nivelul 3:

- pentru tranziția nivel 0 → nivel 1, are de atins pragul de $250 + 0 * 50 = 250$ XP
- pentru tranziția nivel 1 → nivel 2, are de atins pragul de $250 + 1 * 50 = 300$ XP
- pentru tranziția nivel 2 → nivel 3, are de atins pragul de $250 + 2 * 50 = 350$ XP
- pentru tranziția nivel 3 → nivel 4, are de atins pragul de $250 + 3 * 50 = 400$ XP, iar cu 360 XP se va opri la nivelul 3.

Observație: La avansarea în nivel a unui erou, acesta va reveni la 100% HP.

Abilități

Fiecare tip de erou are un anumit set de abilități, ai căror parametri sau ale căror efecte depind de **terenul pe care se desfășoară luptă (land modifiers)** și de **personajul asupra căruia acționează (race modifier)** (**Atenție! Amplificatorii sunt multiplicativi, indiferent dacă abilitatea dă flat damage sau procent din stats-urile adversarului!**) Mai exact, mai mulți aplicatori sunt legați în calcule prin operația de înmulțire, și nu prin operația de adunare.

Observație: La nivelul 0 toate abilitățile au doar efectul de bază!

Detaliem în continuare.

Pyromancer

Abil în manevrarea focului.

HP: 500 initial, +50 per nivel.

Fireblast - damage mare în runda curentă.

- Damage: 350, +50/level

Victimă	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Ignite - damage în runda curentă + damage mai mic în următoarele 2 runde.

- base damage: 150, +20/level
- 50 damage per rundă, +30/level.

Modificatorii de mai jos se aplică atât pentru base damage, cât și pentru damage periodic:

Victimă	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%

Wizard	+5%
--------	-----

Pe terenul de tip Volcanic Pyromancer-ul se hrănește cu energia mediului înconjurător iar abilitățile sale dau cu 25% mai mult damage.

Knight

HP: 900 initial, +80/level.

Execute - damage în runda curentă sau, dacă adversarul are un număr de HP mai mic decât o anumită limită, va fi ucis instantaneu (**damage-ul dat este egal cu HP-ul adversarului**).

- base damage:200, +30/level
- HP limit: 20% * viața teoretic maximă a victimei la nivelul ei; +1% /level, până la un maxim de 40%

Damage-ul (nu și limita de viață) se modifică în funcție de victimă ca mai jos:

Victimă	Modificator
Rogue	+15%
Knight	+0%
Pyromancer	+10%
Wizard	-20%

Slam - damage + incapacitarea adversarului (imposibilitate de mișcare) pentru următoarea rundă.

- base damage: 100 base damage. +40 /level

Victimă	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Knight-ul este expert în lupta corp la corp. Pe terenul de tip land el este mai puternic decât celelalte clase fiind specializat pe lupta în câmp deschis și primește 15% bonus damage.

Wizard

HP: 400 initial, +30 per nivel.

Wizard-ul are o capacitate mentală superioară care îi permite să reziste în mediul de deșert prin meditație și îmbunătățirea abilităților. Pe terenul de tip **Desert** abilitățile sale sunt cu **10%** mai puternice.

Drain - scade din viața adversarului proporțional cu cât are deja.

- procent: 20%, +5% /level
- HP de bază: $\min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$
- $\Rightarrow \text{damage} = \text{procent} * \min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$

Modificatorii de mai jos se aplică asupra variabilei procent. Ei sunt multiplicativi, i.e. dacă de exemplu avem un Wizard level 4 ($\Rightarrow 20\% + 5\% * 4 = 40\%$) și el aplică Drain asupra unui Rogue, avem un procent total de $40\% - (20\% \text{ din } 40\%) = 40\% - 8\% = 32\%$, deci formula de calcul de damage devine $\text{damage} = 0.32 * \min(0.3 * \text{OPPONENT_MAX_HP}, \text{OPPONENT_CURRENT_HP})$.

Victimă	Modificator
Rogue	-20%
Knight	+20%
Pyromancer	-10%
Wizard	+5%

Deflect - dă damage egal cu un procent din damage-ul total (*fără race modifiers*) pe care îl primește de la adversar

- procent: 35%, +2% /level, până la un maxim de 70%
- nu are niciun efect asupra unui Wizard (doi eroi de tip Wizard nu își dau reciproc/recursiv damage)

Victimă	Modificator
Rogue	+20%
Knight	+40%
Pyromancer	+30%
Wizard	N/A

Rogue

Expert în sneak-attacks.

HP : 600 initial, +40 /level

Backstab - damage în runda curentă, cu posibilitate de critical hit.

- base damage: 200, +20 dmg /level.
- O dată la 3 lovituri (lovitură = aplicat Backstab, pe orice teren) Rogue-ul poate da **1.5x** damage, **doar dacă în acel moment se afla pe terenul de tip Woods**, altfel se reia ciclul.

Victimă	Modificator
Rogue	+20%
Knight	-10%
Pyromancer	+25%
Wizard	+25%

Paralysis - damage prelungit + incapacitarea adversarului pentru un număr de runde

- damage/rundă: 40, +10/level - se aplică în runda curentă (în care se desfășoară lupta) + rundele extra
- număr de runde overtime: 3 (6, dacă lupta se desfășoară pe teren Woods)

Victimă	Modificator damage

Rogue	-10%
Knight	-20%
Pyromancer	+20%
Wizard	+25%

Datorită abilităților sale de camuflaj Rogue-ul este mai puternic pe terenul de tip pădure (**Woods**). Pe acest tip de teren Rogue-ul primește **15%** bonus damage.

Mecanism de joc

Jocul este bazat pe runde. Într-o rundă toate personajele execută câte o mișcare (bine determinată), iar dacă două ajung în aceeași locație, se luptă.

Observație: mutarea caracterelor se execută simultan; așadar, doi eroi se presupun a fi ajuns în aceeași locație doar dacă ei se află în același loc pe hartă **după executarea mișcării tuturor eroilor**.

O luptă funcționează astfel: fiecare erou își va calcula parametrii propriilor abilități, în funcție de nivelul lor și de terenul pe care se află. Apoi fiecare abilitate va fi modificată în funcție de victimă. După aplicarea abilităților, eroii își vor calcula noile HP și XP după caz (XP-ul se va modifica doar dacă eroul a câștigat, respectiv dacă și-a omorât adversarul), după care runda se încheie. **Atenție, în cazul în care doi eroi se întâlnesc pe aceeași poziție, se calculează în primul rând damage-ul primit de ce doi de la abilitățile overtime care îi afectează. Dacă unul din ei moare din cauza unei abilități overtime, lupta nu va mai avea loc.**

Mențiuni

- Harta nu este circulară și nu veți întâmpina cazuri în care eroul să fie nevoit să iasă din hartă.
- Nu vor exista cazuri în care să se lupte mai mult de două personaje în același loc.
- Incapacitatea înseamnă doar imposibilitatea de mișcare. Campionii aflați sub influența incapacității pot în continuare să își folosească abilitățile.
- La abilitățile cu efect prelungit (pe mai multe runde), dacă un personaj se află deja sub efectul unei abilități prelungite și suferă de la o nouă abilitate cu efect prelungit, noul efect îl va înlocui pe cel vechi; dacă, de exemplu, un erou mai are de suferit 5 runde de pe urma unui Paralysis și un Warrior îi aplică un Slam, atunci imobilizarea lui va dura doar o rundă și nu va mai primi damage pe rundele următoare. De asemenea, damage-ul unei abilitati overtime este calculat în funcție de level-ul și terenul unde a avut loc batalia și nu se modifică dacă cel afectat se mută pe un alt tip de teren sau dacă adversarul face level up.
- Un campion nu primește experiență dacă omoară un adversar folosind o abilitate cu efect prelungit.
- Abilitățile cu efect prelungit țin cont de terenul pe care a avut loc lupta.
- Pentru o implemmentare mai ușoară, se va presupune că toate abilitățile se dau simultan, iar acestea țin cont de parametrii inițiali (de la începutul luptei) ai eroilor (HP, level).

Aproximări și rotunjiri

- Damage-ul se calculează cu rotunjire la `int`; folosiți `Math.round`.
- **Atenție, pentru a nu avea probleme din cauza rotunjirii, vă sfătuim să folosiți `Math.round` pentru fiecare abilitate în parte, imediat după ce ați aplicat amplificările datorate terenului și race-ului adversarului.** Acest lucru este foarte important, fapt demonstrat de următorul caz: prima abilitate oferă un damage de `111.5`, iar cea de-a doua abilitate oferă un damage de `111.6`:

1. Dacă se face rotunjire după fiecare abilitate, primul damage va fi de 112, iar al doilea va fi tot 112, damage-ul total fiind `112 + 112 = 224` (cel corect, găsit în teste).
2. Dacă se face rotunjire la final, după cumularea damage-urilor din cele două abilități (`111.5 + 111.6 = 223.1`), rezultatul total va fi 223, diferit de cel din teste. Vă rugăm să acordați atenție acestui aspect pentru a evita eventualele greșeli.

- Pentru a nu avea probleme cu aproximările, vă rugăm să folosiți doar variabile de tip `float` atunci când aveți nevoie să rețineți numere cu virgulă. Cu alte cuvinte, declarați constantele cu virgulă folosind "f" la final (de exemplu, utilizați `1.3f`, și nu `1.3`).

Cerințe

Ne dorim să modelăm acest joc 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 eroilor noștri.

În soluțiile voastre, entry-point-ul (metoda `public static void main(String[] args)`) va fi într-o clasă numită `Main` dintr-un pachet numit `main`. Primul argument este numele fișierului de intrare, al doilea este numele fișierului de ieșire. Nu schimbați numele clasei `Main`, numele pachetului `main` sau ordinea argumentelor.

Double Dispatch

Pornind de la avantajele verificării dinamice a tipurilor (la runtime), trebuie să folosiți în cadrul acestei etape a proiectului conceptul de `Double Dispatch`. 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.

Input

Pe prima linie a fișierului se află 2 numere `N` și `M` reprezentând dimensiunile terenului. Pe următoarele `N` linii se vor găsi `N` string-uri de lungime `M` fiecare, cu litere corespunzătoare numelor terenurilor (`W,L,V,D`).

Pe următoarea linie se va găsi un număr `P` reprezentând numărul de personaje. Pe următoarele `P` linii se vor găsi câte un string reprezentând rasa fiecărui personaj și poziția lui inițială (e.g. "W 0 1" înseamnă un Wizard poziționat pe rândul 0 coloana 1).

Pe următoarea linie se găsește `R`, numărul de runde. Fiecare rundă e descrisă de `P` litere, indicând direcția în care se mișcă fiecare personaj. Acestea pot lua una din următoarele valori:

- 'U'(up ⇔ rând-)
- 'D'(down ⇔ rând++)
- 'L'(left ⇔ col-)
- 'R'(right ⇔ col++)
- '_' (fără mișcare)

Direcțiile sunt atribuite eroilor în ordinea în care au fost descrise personajele în fișierul de intrare.

Output

În fișierul de ieșire veți adăuga `P` linii în formatul `rasa_pers level_pers xp_pers hp_pers row col`, câte una pentru fiecare erou, în ordinea în care au fost "declarați" inițial în fișierul de intrare. Dacă personajul este mort, în loc de statistici, pe linia corespunzătoare lui, puneți "dead" (e.g. `K dead` înseamnă un Knight mort).

Exemplu input/output

Exemplu Input:

```
1 1
L
2
```



```
W 0 0
R 0 0
2
—
—
```

Output asteptat:

```
W dead
R 0 200 340 0 0
```

Explicatie:

- Runda 1:

Damage dat de Rogue:

```
Backstab: 200
Amplificator de teren: 1.0
Amplificator de rasa : 1.25
Total: 250
```

```
Paralysis: 40
Amplificator de teren: 1.0
Amplificator de rasa : 1.25
Total: 50
```

Total damage Rogue: 300

Damage dat de Wizard:

```
Drain: 36
Amplificator de teren: 1.0
Amplificator de rasa : 0.8
Total: 28.8 ~= 29
```

Deflect:

```
Damage Backstab fara modificatori de rasa: 200
Damage Paralysis fara modificatori de rasa: 40
Damage Deflect: (200 + 40) * 0.35 = 84
Amplificator de teren: 1.0
Amplificator de rasa : 1.2
Total: 100.8 ~= 101
Damage total Wizard: 29 + 101 = 130
```

```
Hp Rogue la finalul primei runde: 470
Hp Wizard la finalul primei runde: 100
```

```
Hp Rogue la inceputul primei runde (dupa aplicarea DoT): 470
Hp Wizard la inceputul primei runde (dupa aplicarea DoT): 50
```

- Runda 2

este asemanatoare cu prima runda, iar rezultatul va fi:

Hp Rogue la finalul a doua runde: 340

Hp Wizard la finalul a doua runde: 0

API

Vă punem la dispoziție un **API** care vă permite lucrul cu fișiere. `FileIO` este clasa pe care o veți instanția pentru fiecare interacțiune cu fișierele.




Pentru includerea acestei clase în proiect consultați secțiunea **Utile**, aflată mai jos.

De asemenea, metodele din clasa `String` (`split`, `charAt`, `length`, `toCharArray` etc) vă pot fi de mare ajutor. Consultați documentația clasei `String` (link în secțiunea **Utile**).

Indicații

- separați conceptele de sine stătătoare în clase separate, nu le îmbinați
- adaptați agregarea și moștenirea la situație (abilitățile, eroii etc), grupați pe cât posibil informația și acțiunile comune în clase generale
- deduceți semnăturile metodelor așa încât să se poată aplica pe orice tip de erou, respectiv tip de locație, abilități etc
- încercați abordări din unghiuri diferite; de exemplu, un erou poate aplica o abilitate asupra altuia, dar ne putem gândi și că o abilitate se poate aplica asupra unui erou (eliminând astfel agentul), sau că un erou poate suferi de la o abilitate etc
- *folosiți-vă de abilitatea runtime-ului Java de a executa metodele din clasele derivate*
- **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
- **Atenție!** Ar fi util să aveți în vedere utilizarea unui *factory* pentru crearea eroilor și a unei instanțe *singleton* pentru hartă; aceste două lucruri **NU** sunt obligatorii în cadrul acestei etape, **NU** vor exista depunctări dacă nu sunt prezente, dar ar fi în avantajul vostru să le aveți în implementare.

Git

Va sfatuim să folosiți git pentru versionarea codului. Puteți utiliza platforma pusă la dispoziție de facultate ( [Gitlab](#), utilizând conturile de `cs.curs`) sau alte soluții externe ( [Bitbucket](#),  [Github](#)). Pentru a primi bonusul pentru utilizarea git, fiecare commit ar trebui să reprezinte un feature al temei (asta înseamnă că un singur commit nu este de ajuns, dar nici nu ar trebui să existe commit-uri pentru fiecare clasă pe care o adăugați). De asemenea, la încărcarea temei, **trebuie să adăugați folder .git** în arhivă.



Atentie!








Repository-urile, indiferent de platforma pe care o alegeți, **trebuie să fie private !!!**


Punctaj

- 60p trecerea testelor
- 20p folosirea Double Dispatch
- 20p **coding style** (vezi [checkstyle](#))
- 15p design și organizare
- 10p README clar, concis
- 20p BONUS pentru folosirea GIT
- TOTAL: 145p

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**)

Dacă numărul de erori depistate de testele de  [Checkstyle](#) [4] depășește o treime din punctajul maxim, atunci punctele pentru coding-style nu vor fi acordate iar dacă punctajul este negativ, *acesta se trunchiază la 0*.

Exemple:



- `punctaj_total = 100` și `nr_erori = 200` \Rightarrow `nota_finala = 80`
- `punctaj_total = 100` și `nr_erori = 29` \Rightarrow `nota_finala = 100`
- `punctaj_total = 80` și `nr_erori = 30` \Rightarrow `nota_finala = 80`
- `punctaj_total = 80` și `nr_erori = 31` \Rightarrow `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.

 [You shall indeed not pass !!](#)

Structura arhivei






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

- fișierul README
- fișierul `main/Main.java` (entry-point-ul aplicației voastre care se află în pachetul `main`)
- alte fișiere **organizate** cu implementarea **voastră**



Nu încărcați fișierele de test, checker-ul sau documente generate cu JavaDoc.

Link-uri utile

-  [Parametri în linia de comandă în eclipse](#)
-  [Documentație clasa String](#)
-  [Validarea automată a codului în Java](#)
-  [Clean code](#)
- [Tutorial checkstyle](#)
- [Double Dispatch - scurt tutorial](#)
- [Indicații pentru teme](#)
-  [Checker repo](#)