README – Tema POO - WORLD OF MARCEL

Borbei Andreea – grupa 323CC
Nivel de dificultate: mediu
Timp alocat rezolvarii: 2 saptamani

Implementare:

Clasa Game:

Clasa Game are 2 membri: o lista de obiecte de clasa Account si un dictionar al carui cheie este tipul celulei, iar valoarea este o lista de stringuri (povestile ce se vor afisa pentru fiecare casuta).
Instantierea clasei se realizeaza cu ajutorul sablonului **Singleton** cu instantiere intarziata pentru a reduce numarul de instante ale clasei. In acest sens am declarat o instanta statica a clasei, am declarat constructorul clasei privat pentru a nu putea fi accesat din exteriorul clasei si am implementat metoda getInstance care verifica daca instanta declarata a mai fost instantiata sau nu. Daca a mai fost, aceasta este returnata de metoda, altfel se instantiaza folosind constructorul si se returneaza. Astfel exista mereu o singura instanta a clasei Game.

Am implementat **metoda run** care la randul ei apeleaza metodele readAccountsJSON si readStoriesJSON care parseaza fisierele de intrare JSON si populeaza lista de conturi si dictionarul de povesti. Apoi se apeleaza metoda initializeNewGame.

Metoda initializeNewGame primeste ca parametru un string ce reprezinta modul de joc.

Daca se alege ca **mod de joc terminal**, utilizatorul trebuie sa se logheze cu email-ul si parola. Intai se citeste de la tastatura email-ul. Se parcurge lista de conturi si se verifica daca exista vreun cont ce corespunde adresei de email introduse. Daca se gaseste un cont, se trece mai departe la citirea parolei, altfel utilizatorul trebuie sa introduca un nou email si cautarea se reia. Se citeste de la tastatura si parola si se verifica daca corespunde cu parola contului gasit. Daca nu corespunde, se va afisa un mesaj ce anunta utilizatorul ca parola este gresita si trebuie introdusa din nou parola. Utilizatorul ale 3 sanse sa introduca o parola corecta, altfel jocul se incheie.

Dupa logare, utilizatorul trebuie sa aleaga personajul cu care doreste sa joace. Se apeleaza metoda chooseCharacter care returneaza o referinta la personajul ales de catre utilizator.

Dupa alegerea personajului, se genereaza harta hardcodata si se seteaza celula de pe care porneste personajul(cu coordonatele (0, 0)) ca fiind vizitata. Dupa fiecare mutare se afiseaza harta actualizata (nu se afiseaza harta atunci cand jucatorul este pe o celula SHOP sau ENEMY).

Se incepe parcurgerea scenariului (fiecare mutare sau actiune se realizeaza prin apasarea tastei P). Parcurgerea hartii (si jocul in sine) se opreste atunci cand viata personajului ajunge la 0 sau cand personajul ajunge pe o celula de tip FINISH. Se efectueaza 3 mutari la dreapta. Dupa fiecare mutare se verifica daca se gasesc monede pe casuta (acestea fiind de tip EMPTY - exista o sansa de 20% de a gasi monede pe o celula EMPTY). In urma ultimei mutari la dreapta, personajul ajunge pe o celula SHOP. Se afiseaza continutul magazinului si se cumpara 2 potiuni (se verifica daca potiunea poate fi cumparata apeland functia buyPotion din clasa Character. Daca functia returneaza 1, se apeleaza functia buyPotion din clasa Shop si potiunea se adauga la inventarul personajului).

Se realizeaza o mutare la dreapta, apoi se realizeaza 3 mutari in jos (fiind celule de tip EMPTY, se apeleaza din nou metoda ce verifica daca se gasesc monede in celula). Dupa ultima mutare, personajul

ajunge pe o casuta de tip ENEMY. Se apeleaza metoda fight ce modeleaza lupta dintre personaj si enemy. Daca jucatorul castiga, se verifica celula pentru monede cu ajutorul metodei checkEnemyCellForMoney si se realizeaza ultima mutare in jos (se ajunge pe o celula FINISH si jocul se incheie). Daca jucatorul pierde lupta, jocul se incheie.

Metoda chooseCharacter primeste ca parametru un cont. Parcurge element cu element lista de personaje din contul curent; pentru fiecare personaj, utilizatorul trebuie sa scrie la tastatura "yes" daca alege personajul curent sau "no" daca nu il alege. Daca il alege, personajul este returnat. Daca nu alege nimic, metoda returneaza null.

Metoda checkEmptyCellForMoney primeste ca parametru un personaj. Conform cerintei, exista o sansa de 20% ca un personaj sa gaseasca monede pe o celula de tip EMPTY. Astfel, cu ajutorul unui obiect de tip Random se simuleaza aceasta sansa(se genereaza un numar din intervalul [0, 4], probabilitatea ca numarul ales sa fie 0 este de 1/5 = 0.2). Daca conditia este indeplinita si se gasesc monede, acestea se adauga la inventarul personajului.

Metoda checkEnemyCellForMoney primeste ca parametru un personaj. Conform cerintei, exista o sansa de 80% ca un personaj sa primeasca monede. Similar ca la metoda checkEmptyCellForMoney, se simuleaza probabilitatea(se genereaza un numar din intervalul [0, 4], probabilitatea ca numarul ales sa fie diferit de 0 este de 4/5 = 0.8). Daca conditia este indeplinita, monedele se adauga la inventarul personajului.

Metoda showStory primeste ca parametru o celula. Se verifica daca celula data a fost vizitata. Daca este o celula nevizitata, atunci cu ajutorul unui obiect de clasa Random se genereaza un index intre 0 si (marimea listei de povesti a tipului celulei - 1), se obtine povestea de la indexul respectiv din dictionarul de povesti, avand drept cheie tipul celulei si se returneaza povestea. Daca este celula a fost vizitata, metoda returneaza null.

Metoda fight primeste ca parametru 2 personaje (hero si enemy). Voi folosi o variabila (round) in functie de care ataca hero sau enemy (daca round e para, e randul lui hero sa atace, altfel e randul lui enemy sa atace). Inainte de fiecare atac, se afiseaza date relevante pentru lupta despre hero si despre enemy. Atacul lui hero urmareste scenariul impus de cerinta: se verifica daca acesta are spell-uri ; daca are, acestea se afiseaza si se foloseste una (1 spell/atac); va ataca in acest fel pana la epuizarea spell-urilor; daca nu mai are spell-uri, se verifica daca are potiuni in inventar si se vor folosi (1 potiune/atac) pana la epuizarea lor. Cand nu mai exista nici spell-uri, nici potiuni, hero va ataca normal pana la finalul luptei. Atacul lui enemy poate fi ori cu spell (sansa de 25%), ori atac normal (sansa de 75%). Aceste probabilitati se implementeaza cu ajutorul unui obiect de clasa Random (se genereaza un numar din intervalul [0, 3], probabilitatea ca numarul ales sa fie 0 este de 1/4 = 0.25, astfel daca numarul generat este 0, se ataca cu spell, altfel se ataca normal). Daca se ataca cu spell, se verifica daca enemy are mana suficienta pentru a folosi spell-ul, altfel ataca normal. Daca castiga hero, metoda returneaza 1, altfel returneaza 0.

Metoda readStoriesJSON parseaza fisierul stories.json si populeaza dictionarul de povesti (membrul clasei). Se obtine din fisier un JSONArray, iar fiecare element al array-ului are 2 perechi

„nume/valoare” („type” unde valoarea este tipul celulei si „value” unde valoarea este string-ul ce reprezinta povestea). Din fiecare obiect se iau valorile (tipul si povestea) si se adauga la dictionar (cheia este tipul si valoarea este povestea).

Metoda readAccountsJSON parseaza fisierul accounts.json si populeaza lista de conturi. Se obtine un JSONArray, fiecare obiect din array reprezentand un cont. Se vor lua valorile din fiecare obiect si se vor adauga la un obiect de tip Account care dupa populare, se va adauga la lista de conturi.

Daca se alege **modul de joc GUI**, se porneste o fereastra de log in (obiect de clasa LogInWindow). Dupa logare fereastra de log in deschide o fereastra in care jucatorul isi va alege personajul cu care vrea sa joace. Dupa alegerea personajului, fereastra deschide o fereastra de joc (de clasa MainWindow). Pe parcursul jocului, in functie de celula pe care se afla jucatorul, se pot deschide ferestre de lupta (obiecte de clasa BattleWindow) sau ferestre de magazin (obiecte de clasa ShopWindow). La finalul jocului, se deschide o fereastra (obiect de clasa StatusWindow) ce prezinta progresul personajului in joc.

Clasele noi implementate pentru interfata grafica(toate clasele mostenesc clasa JFrame si implementeaza interfata ActionListener) **:**

Clasa LogInWindow:

Membrii clasei sunt componentele grafice ce vor fi afisate pe fereastra, iar pe langa acestea am considerat necesar un obiect de clasa Game (pentru a avea acces la lista de conturi) si un obiect de clasa Account.

Pentru a seta imaginea de background a ferestrei am folosit un obiect de clasa JLabel (ale carui dimensiuni sunt egale cu dimensiunile ferestrei) si am setat imaginea apeland metoda setIcon.
O alta componenta a ferestrei este logo-ul jocului pe care l-am adaugat in mod similar cu background-ul (dar la dimensiuni mai mici).

Pentru caseta de log in am folosit un obiect de clasa JPanel (cu layout de tip Grid) la care am adaugat doua obiecte JLabel (care vor indica care este pentru email si care este cea pentru parola), un obiect JTextField (pentru introducerea email-ului), un obiect JPasswordField (pentru introducerea parolei – am folosit JPasswordField si nu JTextField, pentru a putea ascunde caracterele din parola), un obiect JLabel care anunta ca email-ul sau parola sunt gresite (obiectul va fi invizibil pana cand se introduc date gresite) si un obiect JButton pe care am adaugat un ActionListener(astfel la apasarea butonului se va apela metoda actionPerformed).

Metoda actionPerformed este apelata atunci cand utilizatorul apasa pe buton si verifica daca email-ul si parola introduse sunt corecte (adica corespund unui cont existent in lista de conturi; pentru asta se apeleaza metoda isExistiongAccount), caz in care fereastra curenta sa distruge si se deschide o noua fereastra in care utilizatorul va alege personajul cu care doreste sa joace (personaj din lista de personaje a contului pe care s-a logat). Daca datele nu sunt corecte, errorMessage (obiect de clasa JLabel) devine vizibil si se afiseaza un mesaj de eroare.

Clasa CharacterWindow:

CharacterWindow are ca membri ai clasei, pe langa elementele grafice ale ferestrei, doua obiecte: unul de clasa Game si unul de clasa Account (ce retine contul pe care s-a logat utilizatorul la fereastra de log in). In aceasta fereastra am construit un „slideshow” ce afiseaza personajele din lista de personaje ale contului. Afisarea personajului urmator se realizeaza prin apasarea butonului next. Afisarea personajului anterior se realizeaza prin apasarea butonului prev.

Acest „slideshow” este alcatuit din cele doua obiecte JButton deja mentionate (next si prev), dintr-un obiect JPanel ce cuprinde cate un label si un textfield pentru fiecare atribut al personajului (nume, profesie, nivel si experienta), un obiect JLabel cu ajutorul caruia am afisat imaginea personajului (cate o imagine diferita pentru fiecare tip de personaj) si un buton play care porneste fereastra de joc. La deschiderea ferestrei este afisat primul personaj din lista.

Am implementat **metodele showNextCharacter()** si **showPrevCharacter()** care afiseaza urmatorul personaj, respectiv personajul anterior din lista si actualizeaza campurile cu datele personajului afisat.

Metoda actionPerformed verifica care buton este sursa evenimentului si in functie de asta, afiseaza urmatorul personaj din lista (apeleaza metoda showNextCharacter daca utilizatorul apasa butonul next), afiseaza personajul anterior din lista de personaje a contului (apeleaza metoda showPrevCharacter daca utilizatorul apasa butonul prev) sau deschide fereastra de joc (daca jucatorul apasa play; jucatorul intra in joc cu ultimul personaj afisat in fereastra).

Clasa MainWindow:

Clasa modeleaza fereastra principala a jocului. Pentru a realiza harta am folosit un panel cu layout Grid. In panel am pus o matrice de label-uri cu icons ce vor afisa un icon corespunzator in functie de elementul din celula (pentru shop, enemy sau daca se gasesc monede pe o celula goala). Sub harta, am adaugat un panel in care am afisat viata curenta a jucatorului si monedele pe care le detine. Sub acest panel am pus un obiect de clasa JTextArea in care voi afisa povestile celulelor nevizitate. In stanga hartii am pus o imagine cu personajul narator care ghideaza jucatorul si un panel in care vor aparea indicatiile acestuia (se afiseaza mereu un text si butoane, in functie de etapa jocului: la inceputul jocului se afiseaza butonul de start, pe parcursul jocului se afiseaza butoanele de deplasare pe harta in functie de miscarile posibile, iar la final se afiseaza butonul de exit).

La apasarea butonului start, jocul porneste si se afiseaza butoanele pentru variantele posibile de deplasari pe harta. Dupa fiecare deplasare, se actualizeaza valorile pentru viata si monedele detinute de jucator, se verifica tipul de celula pe care a ajuns (cu metoda checkCell) si se afiseaza o poveste specifica tipului celulei daca celula este nevizitata. Jocul continua pana cand jucatorul ramane fara viata sau pana cand ajunge pe celula de tip FINISH.

Metoda actionPerformed apeleaza metode in functie de butonul apasat de utilizator. Daca se apasa butonul start se porneste jocul si se afiseaza mutarile disponibile. Daca se apasa butonul exit, fereastra se inchide si se deschide fereastra finala ce afiseaza progresul din joc. Daca se apasa unul din cele 4 butoane de deplasare pe harta, se actualizeaza iconita afisata pe harta de la celula pe care se afla inainte de mutare jucatorul, se face mutarea, se actualizeaza iconita celulei curente (pentru a indica pozitia jucatorului pe harta), apoi se verifica celula.

Metoda showPossibleMove primeste ca parametru un obiect de tip Cell (va primi mereu celula curenta pe care se afla jucatorul) si verifica, in functie de coordonatele celulei, care sunt directiile in care se poate deplasa jucatorul astfel incat sa nu iasa in afara hartii. Daca o mutarea intr-o directie anume este imposibila, butonul corespondent devine inaccesibil utilizatorului.

Am implementat **metoda checkCell** primeste ca parametru mereu celula curenta. Metoda verifica tipul celulei si in functie de tipul ei fie deschide fereastra pentru magazin, fie dechide o fereastra de battle. Daca celula este de tip FINISH, jocul este castigat(butoanele de mutari devin inaccesibile jucatorului si devine vizibil butonul de exit).

*pentru ca labelul ce afiseaza discursul personajului narator sa se modifice(si ca butoanele de deplasare sa devina inaccesibile trebuie sa se apese orice buton de deplasare; deplasarea nu se va realiza, dar este necesar acest lucru deoarece editarea labelul se face in metoda actionPerformed)

Clasa ShopWindow:

Clasa implementeaza un slideshow asemanator cu cel de la clasa CharacterWindow. Jucatorul poate vedea cate potiuni sunt in magazin (numar afisat in promptul ferestrei), cantitatea de monede pe care o are si poate cumpara o potiune apasand butonul buyButton care are afisat si pretul potiunii. Daca jucatorul cumpara toate potiunile din magazin, fereastra se inchide singura.

Metodele showNextPotion si showPrevPotion au o implementare asemanatoare cu cele din clasa CharacterWindow.

Clasa BattleWindow:

Clasa BattleWindow extinde JFrame si implementeaza ActionListener. Fereastra va contine imaginile cu cei doi luptatori(hero si enemy), iar sub fiecare imagine va fi afisat cate un panel ce contine informatii relevante bataliei despre cei doi (hero – se afiseaza viata si mana curenta, nivelul si experienta, iar pentru enemy se afiseaza viata, mana si proectiile la cele 3 spell-uri). In josul ferestrei se va afisa meniul din care jucatorul va avea posibilitatea de a alege modalitatea de atac (atac normal sau atac cu spell) sau potiunea pe care doreste sa o foloseasca.

Lupta continua pana cand unul din cei doi ramane fara viata. Cand unul ramane fara viata, este afisat butonul care arata rezultatul bataliei, iar dupa apasarea acestuia fereastra se inchide.

Metoda actionPerformed verifica ce buton a fost apasat de catre utilizator si se realizeaza actiunile specifice apasarii butonului respectiv (butonul de atac normal initiaza un atac normal asupra inamicului; butoanele pentru spell-uri verifica daca se poate utiliza spell-ul dorit, altfel eroul ataca normal; butoanele pentru potiuni folosesc potiunea dorita; de asemenea, in toate cazurile, se actualizeaza labelurile ce sufera modificari in urma atacului, atat la hero, cat si la enemy, sau in urma folosirii potiunii). Dupa actiunea jucatorului, urmeaza atacul inamicului (se apeleaza metoda enemyAttack). Daca se apasa butonul victoryStatus, inseamna ca batalia s-a incheiat si se inchide fereastra.

Metoda enemyAttack() descrie felul in care va ataca inamicul. Se va simula acea sansa de 25% de atac cu spell. Daca conditia pentru atac cu spell nu este indeplinita, inamicul ataca normal. Daca se foloseste un spell, se actualizeaza labelul ce afiseaza mana inamicului. Indiferent de tipul de atac, se actualizeaza labelul ce afiseaza viata eroului.

*pentru ca butonul victoryStatus sa apara trebuie sa se mai apase o data orice buton

Clasa StatusWindow:

Este o clasa simpla ce impementeaza ActionListener (asculta butonul exitButton) si mosteneste JFrame. Fereastra afiseaza progresul personajului la finalul jocului. La apasarea butonului de exit, fereastra se inchide.

Metoda getPicture este o metoda folosita in toate clasele folosite la interfata grafica pentru a scala imagini la dimensiunile dorite si pentru a transforma un fisier ce contine o imagine intr-un obiect de tip ImageIcon. Metoda citeste un fisier ce contine o imagine, apoi scaleaza la dimensiunile dorite (trimise ca parametrii) obiectul obtinut in urma citirii. Se creeaza un obiect de clasa ImageIcon de la obiectul de clasa Image scalat si se returneza.

Clasele impuse in cerinta :

Clasele Account si Information:

Clasa **Account** are, pe langa membrii clasei, o **clasa interna Information** ce retine credentialele unui jucator, numele, tara de provenienta si o lista de string-uri ce va memora jocurile preferate ale userului.

Instantierea unui obiect de clasa Information se va realiza prin intermediul **sablonului Builder**, ce are rolul de a contrui membru cu membru un obiect (astfel se evita existenta mai multor constructori pentru aceeasi clasa).

In constructorul clasei Information, membrii se instantiaza prin intermediul obiectului de clasa InformationBuilder (astfel se garanteaza ca cel putin membrii credentials si name sunt instantiati).
In interiorul clasei Information construiesc clasa interna InformationBuilder (ce va avea aceeasi membrii ca si Information) ce implementeaza design pattern-ul mentionat. In aceasta clasa, constructorul instantiaza membrii credentials si name fara de care un obiect Information nu poate exista. Metodele country si favouriteGames seteaza membrii cu acelasi nume si returneaza obiectul curent de tip InformationBuilder.

Metoda build verifica daca membrii credentials si name sunt intializati si daca nu sunt, este aruncata o exceptie InformationIncompleteException, altfel returneaza un obiect de tip Information.

Restul metodelor din clasa Information sunt metode getter pentru membrii, deoarece acestia sunt privati(deci nu pot fi accesati din afara claselor Account si Information).

Clasa Credentials:

Clasa Credentials are 2 membri (email si parola - datele cu care utilizatorii se vor loga) ai caror modificatori de acces sunt privati, pentru a respecta principiul incapsularii. Pentru a accesa membrii clasei am definit 4 metode(2 setteri pentru setarea valorilor si 2 getteri pentru a returna valorile). Cei doi membri ai clasei sunt accesibili in afara clasei Credentials doar prin intermediul acestor metode.

Clasa Grid:

Clasa Grid extinde clasa ArrayList tipizata cu clasa ArrayList(care este si ea tipizata cu clasa Cell) pentru a modela cu usurinta harta ce va fi sub forma unei matrici.

In aceasta clasa, pe langa cele 4 metode de deplasare a personajului pe harta, am implementat 2 metode(2 metode statice de generare harti - generarea hartii hardcodate necesara scenariului de testare si generarea unei harti randomizate necesara rularii jocului din interfata grafica) .

Prin cele doua metode de generare se si instantiaza obiectul de tip grid in afara clasei, deoarece cinstructorul clasei este privat si nu poate fi accesat din exteriorul clasei.

Metodele de deplasare pe harta sunt:

1)Metoda goNorth() - verifica daca coordonata x a celulei este mai mare decat 0, caz in care deplasarea se poate realiza, altfel se afiseaza un mesaj. La deplasare, celula curenta este marcata drept vizitata, apoi celula curenta primeste o referinta catre noua celula curenta (cu coordonatele (x - 1, y)).

2)Metoda goSouth() - verifica daca coordonata x a celulei este mai mica decat height, caz in care deplasarea se poate realiza, altfel se afiseaza un mesaj. La deplasare, celula curenta este marcata drept vizitata, apoi celula curenta primeste o referinta catre noua celula curenta (cu coordonatele (x + 1, y)).

3)Metoda goWest() - se verifica daca coordonata y a celulei este mai mare decat 0, caz in care deplasarea se poate realiza, altfel se afiseaza un mesaj. La deplasare, celula curenta este marcata drept vizitata, apoi celula curenta primeste o referinta catre noua celula curenta (cu coordonatele (x, y - 1)).

4)Metoda goEast() - se verifica daca coordonata y a celulei este mai mica decat width, caz in care deplasarea se poate realiza, altfel se afiseaza un mesaj. La deplasare, celula curenta este marcata drept vizitata, apoi celula curenta primeste o referinta catre noua celula curenta (cu coordonatele (x, y + 1)).

Metoda createTestMap() genereaza harta hardcodata necesara scenariului de testare, conform indicatiilor si returneaza obiectul de clasa Grid nou instantiat.

Metoda createMap() genereaza in mod aleator o harta. Pentru fiecare tip de casuta(SHOP si ENEMY) se genereaza cu ajutorul unui obiect de clasa Random, un numar de casute ce vor fi instantiate cu tipul respectiv. Apoi, pentru fiecare celula de tipul respectiv se genereaza coordonatele x si y. Daca se genereaza coordonatele unei casute deja setate, se genereaza alte coordonate, altfel se obtine celula de la coordonatele generate si se seteaza membrul de tip CellElement.

Dupa ce au fost create celulele de tip SHOP si ENEMY, se parcurge intregul grid si toate casutele ce au membrul element null devin de tip EMPTY. La final, obiectul de tip Grid este returnat.

Am considerat necesara suprascrierea metodei toString pentru afisarea unui obiect de clasa Grid sub forma unei matrici. Se parcurge grid-ul. Pentru celula ce coincide cu celula curenta pe care se afla personajul se afiseaza "P", altfel se verifica daca casuta a fost vizitata sau nu. Daca a fost vizitata, se afiseaza caracterul returnat de metoda toCharacter(), daca nu a fost vizitata, se afiseaza "?".

Clasa Cell:

Clasa Cell modeleaza o celula din harta. In aceasta clasa sunt implementate 3 metode:

1) **isVisited()** care verifica daca o casuta a fost vizitata deja sau nu; daca a fost returneaza true, altfel returneaza false. Aceasta metoda este utilizata in timpul rularii jocului, atunci cand se efectueaza o miscare (daca casuta a mai fost vizitata si casuta contine un enemy, personajul nu se mai lupta cu inamicul; de asemenea, de pe casutele empty deja vizitate nu se mai pot colecta monede).

2) **setCoordinates** care seteaza valorile coordonatelor x si y de pe harta

3) **setElement** primeste ca parametru tipul de element al celulei si il seteaza. Pentru a facilita utilizarea metodei toChar pentru orice casuta din harta, am implementat inca 2 clase(Empty si Finish) ce vor implementa interfata CellElement, deci si metoda toChar. Acest lucru este necesar pentru afisarea facila a hartii in rularea jocului din terminal.

Clasa Entity:

Clasa abstracta Entity implementeaza interfata Element (elementul ce este vizitat de visitor; necesara pentru implementarea design pattern-ului Visitor).

Clasa implementeaza metodele **lifeRegeneration(int)** si **manaRegeneration(int)**, ale caror implementari sunt identice (se adauga valoarea primita ca parametru la valoarea curenta a vietii, respectiv a manei si se verifica daca se depaseste sau nu maximul). De asemenea, se implementeaza metoda accept(Visitor) prin care entitatea accepta vizita unui visitor(un spell).

Metoda **useSpell** verifica daca spell-ul primit ca parametru poate fi utilizat (se verifica daca personajul are mana suficienta), apoi enemy-ul(entitatea trimisa prin parametru, nu e neaparat de clasa Enemy) apeleaza metoda accept.

Am implementat o **metoda showAbilties()** care parcurge lista de abilitiati si formateaza un string sub forma unei liste. Aceasta functie este utila la rularea din terminal, atunci cand personajul este in lupta cu inamicul si alege sa foloseasca o abilitate. Atunci aceasta metoda este apelata pentru a se afisa toate abilitatile de care acesta dispune.

Clasa Character:

Clasa abstracta ce mosteneste metodele si membrii clasei Entity. Pe langa acestea, clasa introduce si ea membri precum numele personajului, coordonatele sale pe harta, un obiect de clasa Inventory, experienta si nivelul personajului. Pe langa acestea, se introduc 3 valori pentru atributele strength, charisma si dexterity, care vor influenta damage-ul dat si primit de un personaj.

Am implementat o metoda de cumparare a unei potiuni, in care se verifica daca exista suficienti bani si suficient spatiu in inventar pentru potiuni. Daca exista, potiunea este cumparata(deci numarul de monede si greutatea disponibila a inventarului scad) si metoda returneaza 1, altfel returneaza 0.

Clasele Warrior, Rogue si Mage:

Clasele Warrior, Mage si Rogue mostenesc clasa Character (deci trebuie sa se implementeze metodele abstracte din clasa). Pentru fiecare clasa se seteaza valorile boolene ale protectiilor la spell-uri conform cerintei si apoi, in functie de nivel si de care e atributul principal al clasei, se calculeaza valorile celor 3 atribute (charisma, strength si dexterity). Se instantiaza obiectul de clasa Inventory cu o greutate de inventar (specifica fiecarei clase) si cu o cantitate de monede (egala pentru toate clasele).

Pentru ca nu se specifica in cerinta, am decis ca fiecare personaj va avea 4 abilitati alese in mod aleator. Se genereaza pentru fiecare obiect din lista de abilitati, cu ajutorul unui obiect de clasa Random, valori intregi din intervalul [1, 3]. Daca valoarea generata este 1, se adauga spell de clasa Ice, daca este 2, spell de clasa Fire, respectiv daca este 3, spell de clasa Earth.

Pentru instantierea obiectelor din aceasta clasa se foloseste **design pattern-ul Factory**, motiv pentru care am construit o clasa CharacterFactory ce are o singura metoda getCharacter ce primeste parametrii necesari instantierii unui obiect de clasa Character si pe langa acestea, primeste un parametru String ce contine profesia obiectului ce trebuie instantiat. Astfel, in functie de profesie, se instantiaza un obiect de clasa Warrior, Rogue sau Mage. Obiectul instantiat este returnat. Daca profesia primita ca parametru nu se potriveste cu niciuna din cele 3 mentionate, metoda returneaza null.

Se implementeaza pentru fiecare clasa **metoda getDamage()** si **metoda receiveDamage(int)** mostenite de la clasa Character(damage-ul primit si dat depinde de atributele clasei). Cu cat nivelul unui personaj este mai mare, cu atat valorile atributelor sunt mai mari, deci damage-ul dat creste odata cu cresterea nivelului si damage-ul primit scade odata cu cresterea nivelului.

Clasele Ice, Fire si Earth:

Clasele Ice, Fire si Earth mostenesc clasa Spell (care implementeaza interfata Visitor). Astfel, cele 3 clase trebuie sa implementeze **metoda visit**, ce defineste efectul spell-urilor asupra unei entitati. Asadar, pentru fiecare implementare, se verifica daca entitatea are protectie la spell-ul respectiv;
daca are, damage-ul primit se micsoreaza, daca nu are, primeste intregul damage pe care il poate da spell-ul.

Clasa Enemy:

Clasa Enemy mosteneste clasa abstracta Entity si implementeaza interfata CellElement.

Generarea valorilor pentru currentRemainingLife si currentRemainingMana se realizeaza cu ajutorul unui obiect de clasa Random. Acest obiect va apela metoda nextInt() ce returneaza un intreg cu valori intre lowerbound(inclusiv) si upperbound(exclusiv). Tot cu ajutorul obiectului de clasa Random, care acum apeleaza metoda nextBoolean(), se genereaza aleator valorile de tip boolean pentru protectiile la cele trei spell-uri. Am decis ca fiecare enemy sa aiba 3 abilitati si am generat in mod similar 3 valori cuprinse in intervalul [1, 4) astfel: se genereaza 1 - enemy primeste Ice spell, 2 - enemy primeste Fire spell, respectiv 3 pentru Earth spell. Fiecare spell se adauga la lista de abilitati a entitatii.

Metoda receiveDamage(int) primeste o valoare intreaga drept damage-ul primit de enemy. Exista 2 posibilitati: sa primeasca damage sau sa il evite (sansa de 50%). Daca se genereaza valoarea 1, enemy primeste damage si valoarea vietii sale scade cu valoarea damage-ului (se verifica daca damage-ul este mai mare decat viata, caz in care viata devine 0; se evita aparitia unei valori negative pentru viata). La generarea valorii 0, damage-ul este evitat(nu este afectata valoarea vietii entitatii).

Similar, **metoda getDamage()** simuleaza cu ajutorul unui obiect de clasa Random sansa de 50% de a da damage dublu. Se genereaza 0 sau 1. Pentru 0, enemy da damage normal(valoare fixa, 20), iar pentru 1, damage dublu. Metoda returneaza valoarea damage-ului dat de enemy.

Clasa Inventory:

Clasa Inventory are drept membri o lista tipizata cu obiecte de tip Potion si 2 valori de tip int ce simbolizeaza greutatea maxima a inventarului si numarul de monede pe care le detine entitatea.

Pe langa metodele din cerinta, am implementat o **metoda showPotions()** ce afiseaza potiunile din inventar; aceasta ma va ajuta in cazul in modul de joc ales este din terminal. Atunci cand jucatorul ajunge pe o casuta de tip ENEMY si doreste sa foloseasca o potiune, metoda showPotions() este apelata pentru a afisa jucatorului potiunile de care personajul sau dispune.

Clasele HealthPotion si ManaPotion:

Pentru clasele ManaPotion si HealthPotion am considerat ca este mai util sa folosesc final la membrii claselor, deoarece valorile lor nu sunt modificabile pe parcursul jocului.

Clasele implementeaza interfata Potion, deci si metodele definite in aceasta (**getPrice()** ce intoarce pretul unei potiuni, **getRegeneration()** ce intoarce valoarea regenerarii oferite de potiune si getInventoryWeight() care returneaza greutatea ocupata de potiune in inventar).

Clasa Shop:

Clasa Shop are drept membru o lista de potiuni si implementeaza interfata CellElement.

Populatarea listei de potiuni se face la instantierea unui obiect, in cadrul constructorului.

Un magazin poate avea intre 2 si 4 potiuni, motiv pentru care se genereaza un numar random de potiuni cu ajutorul unui obiect de clasa Random, apeland metoda nextInt(2, 5). Apoi pentru fiecare element al listei se adauga in mod aleator o potiune(se genereaza 1 sau 0; daca se genereaza 1 potiunea va fi de clasa HealthPotion, altfel de clasa ManaPotion).

Am considerat necesara **suprascrierea metodei toString()** pentru a formata afisarea potiunilor din lista. Acest fapt este util atunci cand, la rularea din terminal, personajul ajunge pe o casuta de tip SHOP, deoarece se va afisa un mesaj, apoi lista de potiuni disponibile in magazin pe care jucatorul le poate cumpara.