# Programare orientată pe obiecte

## Tema - Chess

## Partea I

**Data postării: 18.11.2025**
**Ultima actualizare: 24.11.2025, 21:20**

**Deadline: 14.12.2025 ora 23:55**

**Echipă temă: Carmen ODUBĂȘTEANU, Ștefan TURCU, Alexandru TUDOR**

**Facultatea de Automatică şi Calculatoare**

**Universitatea Națională de Știință și Tehnologie Politehnica București**

**Anul universitar 2025 - 2026**

**Seria CC**

## 1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să aplice corect principiile programării orientate pe obiecte studiate în cadrul cursului;
- să construiască o ierarhie de clase, pornind de la un scenariu propus;
- să utilizeze un design orientat-obiect;
- să trateze excepțiile ce pot interveni în timpul rulării unei aplicații;
- să transpună o problemă din viața reală într-o aplicație.

**Șahul** este un joc de strategie pe tablă, o competiție între două persoane. Fiecare jucător controlează o armată de 16 piese, care includ un rege, o regină, doi nebuni, doi cai, două turnuri (ture) și opt pioni. Jocul se desfășoară pe o tablă pătrată de 8x8, formată din 64 de pătrate alternante de culori diferite (alb și negru).

Proiectul propune crearea unui astfel de joc folosind principiile programării orientate pe obiect și funcționalitățile puse la dispoziție de limbajul de programare JAVA, aplicând noțiunile studiate în cadrul orelor de curs și laborator.

## 2 Reguli de joc

Obiectivul principal este de a da șah mat regelui adversarului, adică să-l pui într-o poziție în care este atacat și nu poate scăpa.

Fiecare tip de piesă se mișcă diferit:

- **King**: Se mișcă un pătrat în orice direcție. Noua poziție nu trebuie să fie ocupată de o piesă de aceeași culoare și nu trebuie să reprezinte o amenințare (să nu fie în șah).
- **Queen**: Se mișcă în orice direcție, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Rook**: Se mișcă pe orizontală sau verticală, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Bishop**: Se mișcă pe diagonală, pe oricâte pătrate. Nu poate sări peste alte piese.
- **Knight**: Se mișcă în formă de „L" – două pătrate într-o direcție și apoi un pătrat perpendicular pe aceasta. Calul este singura piesă care poate sări peste alte piese.
- **Pawn**: Se mișcă înainte un pătrat (sau două pătrate la prima mișcare), dar capturează piesele adversarului doar diagonal. Nu poate sări peste alte piese.

Pentru a simplifica jocul, nu vom lua în considerare mutările speciale care există în jocul original. Pentru mai multe detalii legate de regulile jocului, puteți accesa **această adresă**.

Tabla de șah are o structură matriceală, fiind împărțită în 64 de pătrate. Fiecare pătrat este identificat prin combinarea unei litere (de la „A" la „H") pentru coloane și a unui număr (de la 1 la 8) pentru rânduri. Coloanele sunt numerotate de la stânga (A) la dreapta (H), iar rândurile sunt numerotate de jos (1) în sus (8).

Pătratele de pe tabla de șah sunt alternant colorate în alb și negru, începând cu un pătrat de culoare închisă în colțul din stânga jos (A1). Această alternanță de culori este păstrată pe întreaga tablă.

În starea inițială a jocului, fiecare jucător dispune de două rânduri de piese. Primul rând (rândul 1 pentru echipa albă și rândul 8 pentru echipa neagră) include piesele majore: turnurile în colțuri, caii lângă turnuri, nebunii lângă cai, regina pe pătratul de culoarea sa și regele pe pătratul rămas. Al doilea rând (rândul 2 pentru ehipa albă și rândul 7 pentru echipa neagră) este ocupat de pionii fiecărui jucător.

Notația algebrică este folosită pentru a descrie mișcările pieselor pe tabla de șah, identificând fiecare pătrat prin litera coloanei și numărul rândului corespunzător. De exemplu, colțul din stânga jos este „A1", iar colțul din dreapta sus este „H8". Pentru a muta o piesă, spre exemplu un pion, care se află la poziția A2, la poziția A3, se va folosi notația A2-A3.

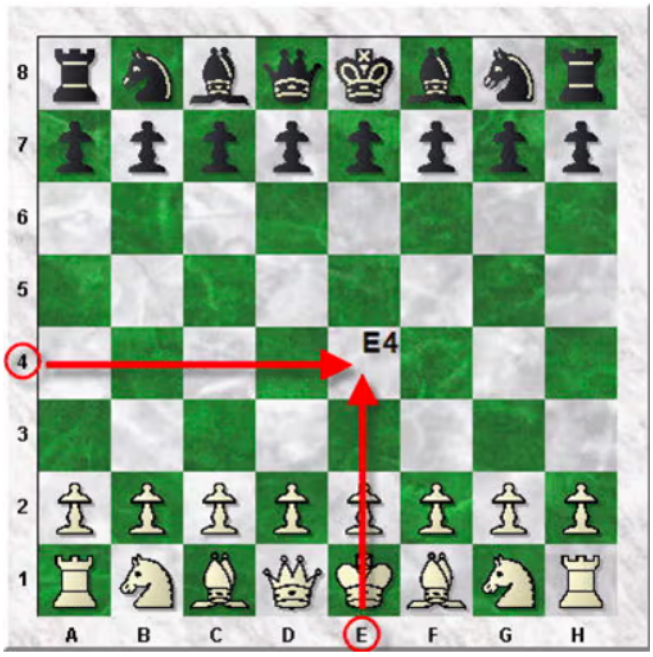


Figura 1: Reprezentarea tablei de șah

# 3 Arhitectura aplicației

## 3.1 CLASE

**Atenție!**

Conținutul claselor este minimal. Puteți adăuga orice alte atribute și metode pe care le considerați folositoare. De asemenea, puteți crea orice alte clase care să vină în ajutor, în raport cu principiile programării orientate pe obiecte.

Metodele menționate trebuie să conțină și să folosească parametrii specificați, însă puteți adăuga alte argumente dacă vă ajută în implementare.

### 3.1.1 Main

Clasa reprezintă punctul de intrare și interfața principală a aplicației. Ea are rolul de a încărca datele din fișiere, de a realiza autentificarea utilizatorilor și de a gestiona meniurile prin care utilizatorul poate porni un joc nou sau poate continua un joc existent. Clasa trebuie să conțină:

- Atribute folosite pentru salvarea datelor preluate din fișierele de intrare, de exemplu:
  - o listă cu toți utilizatorii existenți, de tip `List<User>`;
  - o colecție cu jocurile existente, indexate după identificator, de tip `Map<Integer, Game>`;
  - utilizatorul curent autentificat, de tip `User`.
- `public void read();`

  - citește datele din fișierele de intrare și inițializează colecțiile de utilizatori și jocuri.
- `public void write();`

  - scrie în fișierele JSON starea curentă a utilizatorilor şi a jocurilor (puncte, jocuri noi, jocuri şterse etc.).
- `public User login(String email, String password);`

- caută în colecția internă utilizatorul care are credențialele menționate; dacă autentificarea reușește, setează utilizatorul curent și returnează obiectul `User` corespunzător.

- `public User newAccount(String email, String password);`

  - creează un nou utilizator pe baza datelor primite, îl adaugă în colecția de utilizatori, îl setează ca utilizator curent și returnează obiectul `User` creat.

- `public void run();`

  - pornește flow-ul aplicației: gestionează procesul de autentificare (`login` / `newAccount`), afișează meniul principal și permite utilizatorului să aleagă între începerea unui joc nou, continuarea unui joc existent sau delogare, conform secțiunii *Flow-ul jocului*.

- `public static void main(String[] args);`

  - punctul de intrare în aplicație; creează o instanță a clasei `Main`, apelează metoda `read()` pentru a încărca datele și apoi metoda `run()` pentru a porni interacțiunea cu utilizatorul.

#### 3.1.2 ChessPair

Clasă **generică** `ChessPair<K, V>` ce conține două atribute private de tip `K` (cheie) și `V` (valoare). Obiectele de tip `ChessPair` se vor compara între ele **crescător după cheie** (în cadrul acestei teme, cheile vor fi obiecte de tip `Position`). În plus, clasa trebuie să conțină o metodă care să returneze cheia și valoarea împreună, sub forma unui obiect de tip String. **Clasa trebuie implementată respectând principiul încapsulării!**

#### 3.1.3 Position

Clasa este utilizată pentru a reprezenta coordonatele unui pătrat pe tabla de șah (coloană **x** de tip literă `'A' - 'H'` și linie **y** de tip număr `1 - 8`). Două obiecte de tip `Position` se compară între ele **crescător după coordonata y** (elementul de tip `int`) și, dacă aceasta este egală, **crescător după coordonata x** (elementul de tip `char`).

Clasa conține:

- Coordonata x a poziției pe tablă, de tip **char**.
- Coordonata y a poziției pe tablă, de tip **int**.
- `public boolean equals(Object o);`
  - compară dacă o altă poziție este egală cu această poziție; două poziții sunt considerate egale dacă au aceleași coordonate x și y.
- `public String toString();`
  - returnează o reprezentare sub formă de șir de caractere a poziției (de exemplu `"A2"`).

#### 3.1.4 Piece

Clasă **abstractă** care implementează interfața `ChessPiece` și servește drept clasă de bază pentru toate tipurile de piese din jocul de șah. Clasa gestionează starea comună a pieselor (culoare și poziție) și oferă funcționalități de bază folosite de toate subclasele.

- Culoarea piesei (alb sau negru), de tip **Colors**. Culoarea se setează la crearea piesei (în constructor) și nu se modifică pe parcursul jocului.
- Poziția actuală a piesei pe tablă, de tip **Position**.
- `public Colors getColor();`
  - returnează culoarea piesei.
- `public Position getPosition();`
  - returnează poziția curentă a piesei pe tablă.

- `public void setPosition(Position position);`
  - setează noua poziție a piesei pe tablă.

#### 3.1.5 Clase pentru fiecare tip de piesă

Aceste clase vor extinde clasa `Piece` și vor implementa logica specifică de joc a fiecărui tip de piesă, prin redefinirea metodelor din interfața `ChessPiece`, respectând regulile de mutare corespunzătoare. Clasele sunt **King**, **Queen**, **Rook**, **Bishop**, **Knight**, **Pawn** (acestea pot avea, dacă este nevoie, și atribute suplimentare specifice, cum ar fi informații despre prima mutare a unui pion).

#### 3.1.6 Board

Clasa reprezintă tabla de șah și logica asociată mutărilor. Ea se ocupă cu inițializarea pieselor, validarea și efectuarea mutărilor pe tablă și gestionarea stării pieselor pe parcursul jocului (inclusiv capturile). `Board` este sursa de adevăr pentru ce piesă se află pe fiecare poziție.

- O listă sortată care stochează piesele de șah de pe tablă, asociate cu pozițiile lor, de tip `TreeSet<ChessPair<Position, Piece>>`. Mulțimea este menținută sortată după cheie (`Position`).
- `public void initialize();`
  - inițializează tabla de șah cu piesele la pozițiile inițiale, creând obiectele de tip `Piece` și adăugându-le în lista internă; se asigură că poziția stocată în `Piece` și poziția din `ChessPair` sunt consistente.
- `public void movePiece(Position from, Position to);`
  - verifică mutarea folosind `isValidMove(from, to)` și, dacă mutarea este invalidă, aruncă o `InvalidMoveException`;
  - dacă mutarea este validă, mută piesa de la poziția `from` la poziția `to`: actualizează structura internă (lista de `ChessPair`) și poziția piesei (`piece.setPosition(to)`);
  - dacă la poziția `to` se află o piesă adversă, aceasta este scoasă de pe tablă (pentru a putea fi adăugată ulterior în lista de piese capturate a unui `Player`);
  - dacă piesa mutată este un pion care, în urma mutării, ajunge pe ultima linie a tablei (linia 8), acesta trebuie promovat într-o altă piesă (queen, rook, bishop sau knight), actualizând corespunzător structura internă (înlocuirea pionului cu piesa aleasă în lista de `ChessPair`).
- `public Piece getPieceAt(Position position);`
  - returnează piesa de pe o anumită poziție sau `null` dacă pătratul este liber.
- `public boolean isValidMove(Position from, Position to);`
  - verifică dacă o mutare dată este validă conform regulilor de șah, ţinând cont de: tipul piesei care se mută, configuraţia curentă a tablei (piese proprii şi adverse), limitele tablei şi faptul că, după mutare, regele jucătorului care mută nu rămâne în șah.

#### 3.1.7 Player

Clasa este utilizată pentru a reprezenta un jucător (uman sau computer) implicat într-un joc de șah. Un obiect `Player` reține informații despre identitatea jucătorului, culoarea cu care joacă, piesele pe care le controlează și punctajul său.

- Numele jucătorului, de tip **String**.
- Culoarea pieselor pe care le controlează (alb sau negru), de tip **Colors**.
- Lista de piese capturate de către jucător, de tip **private List<Piece>**.
- Mulțimea sortată de piese disponibile deținute de către jucător (piese aflate pe tablă), de tip **private TreeSet<ChessPair<Position, Piece> >**.
- Numărul total de puncte acumulate pe parcursul jocului curent, de tip **private int**.

- `public void makeMove(Position from, Position to, Board board);`

  - încearcă să efectueze o mutare a unei piese de la poziția `from` la poziția `to` pe tablă;
  - folosește logica din `Board` (de exemplu, apeland `board.isValidMove(...)` sau `board.movePiece(...)`) și, dacă mutarea este invalidă, aruncă o excepție;
  - dacă mutarea este validă și rezultă capturarea unei piese adverse, adaugă piesa în lista de piese capturate și actualizează punctajul jucătorului.

- `public List<Piece> getCapturedPieces();`

  - returnează lista de piese capturate de către jucător.

- `public List<ChessPair<Position, Piece>> getOwnedPieces();`

  - returnează lista de piese deţinute de către jucător (piesele sale aflate pe tablă); lista poate fi menţinută intern şi actualizată la fiecare mutare/captură sau poate fi derivată din `Board`, atâta timp cât este consistentă cu starea tablei.

- `public int getPoints();`

  - returnează punctele jucătorului.

- `public void setPoints(int points);`

  - setează numărul de puncte (de exemplu, dacă jocul s-a finalizat sau dacă o piesă este capturată, conform regulilor de scor).

#### 3.1.8 Game

Clasa coordonează desfășurarea unui joc (meci) de șah. Ea gestionează starea generală a jocului, inclusiv tabla de șah, jucătorii implicați, rândul curent al jucătorului și istoricul mutărilor. `Game` nu mută direct piesele (acest lucru se face prin `Board` și `Player`), ci orchestrează turele și verifică starea jocului (în desfășurare, șah, șah-mat, egalitate).

- Identificatorul jocului, de tip **int**.
- Tabla de șah pe care se desfășoară jocul, de tip **Board**.
- Două obiecte de tip **<Player>** ce reprezintă jucătorul și adversarul său (calculatorul).
- O structură care reține mutările (**obiecte de tip Move**) efectuate pe parcursul jocului, în ordinea executării.
- Indexul jucătorului curent din lista de jucători, de tip **int**.
- `public void start();`

  - începe un joc nou de şah;
  - iniţializează tabla (`board.initialize()`), golește istoricul mutărilor și stabileşte jucătorul care va începe jocul (de exemplu, jucătorul cu piesele albe).

- `public void resume();`

  - reia un joc încărcat din fişiere; nu reiniţializează tabla, ci porneşte de la starea salvată (tabla, mutările şi jucătorul curent fiind deja cunoscute).

- `public void switchPlayer();`

  - trece la următorul jucător, actualizând indexul jucătorului curent în lista de jucători (de obicei comută între cei doi jucători).

- `public boolean checkForCheckMate();`

  - verifică, folosind informațiile din `Board` și piesele jucătorilor, dacă unul dintre jucători este în șah-mat;
  - returnează **true** dacă partida s-a încheiat prin șah-mat și **false** altfel (identitatea jucătorului aflat în șah-mat poate fi stocată în câmpuri suplimentare, dacă este necesar).

- `public void addMove(Player p, Position from, Position to);`

- construiește un **obiect care descrie mutarea** (obiect Move) și îl adaugă în lista de mutări;
- această metodă se apelează după ce mutarea a fost efectuată cu succes la nivel de `Board` / `Player`.

#### 3.1.9 Move

Modelul pentru obiectul care reprezintă o mutare. Acesta conține minim:

- Culoarea jucătorului care a făcut mutarea, de tip **private Colors**;
- Poziția de la care a plecat piesa, de tip **private Position**;
- Poziția la care a ajuns piesa, de tip **private Position**;
- Piesa capturată în urma mutării, de tip **private Piece**, care poate fi `null` dacă nu s-a capturat nimic.

#### 3.1.10 User

Clasa reprezintă un utilizator (cont) care se autentifică în aplicație și are asociate jocurile sale în desfășurare și punctele acumulate în timp. **Clasa trebuie implementată respectând principiul încapsulării!**

- Email-ul utilizatorului, de tip **private String**.
- Parola utilizatorului, de tip **private String**.
- Lista de jocuri asociate utilizatorului și aflate în derulare, de tip **private List<Game>**.
- Numărul total de puncte acumulate pe parcursul jocurilor, de tip **private int**.
- `public void addGame(Game game);`

  - adaugă un joc nou în lista de jocuri ale utilizatorului (de exemplu, un joc nou creat sau un joc încărcat din fișier).
- `public void removeGame(Game game);`

  - șterge un joc din lista de jocuri ale utilizatorului (de exemplu, după ce jocul a fost încheiat sau utilizatorul decide să îl elimine).
- `public List<Game> getActiveGames();`

  - returnează lista de jocuri (în desfășurare) ale utilizatorului.
- `public int getPoints();`

  - returnează punctele totale acumulate de utilizator.
- `public void setPoints(int points);`

  - actualizează numărul total de puncte ale utilizatorului (de exemplu, la finalul unui joc, conform regulilor de scor).

### 3.2 INTERFEȚE

#### 3.2.1 ChessPiece

Interfața `ChessPiece` definește comportamentul comun pentru toate tipurile de piese de șah. Ea conține următoarele metode:

- `List<Position> getPossibleMoves(Board board);`

- returnează o listă de poziții valide unde poate fi mutată piesa, în starea curentă a tablei primită ca argument;
- piesele (cu excepția calului) se opresc în drumul lor când întâlnesc prima piesă a adversarului; în acest caz, piesa adversarului poate fi capturată, iar poziția ei este inclusă în lista de mutări;
- dacă în drumul ei, o piesă (cu excepția calului) întâlnește o altă piesă de aceeași culoare, nu poate sări peste aceasta și nu vor fi incluse poziții dincolo de acea piesă;
- lista întoarsă poate conține mutări care, la nivel de joc, nu sunt permise deoarece ar lăsa propriul rege în șah; această regulă se va verifica în mod obligatoriu în `Board`;
- orice jucător poate muta doar piesele proprii, specifice culorii pe care o deține în joc.

- ```
  boolean checkForCheck(Board board,
                        Position kingPosition);
  ```

  - verifică, folosind starea curentă a tablei, dacă regele aflat la poziția specificată poate primi șah de la piesa curentă;
  - poate fi implementată, de exemplu, verificând dacă `kingPosition` se află în lista întoarsă de `getPossibleMoves(board)`.

- ```
  char type();
  ```

  - returnează caracterul corespunzător tipului de piesă (`'K'` - king, `'Q'` - queen, `'R'` - rook, `'B'` - bishop, `'N'` - knight, `'P'` - pawn).

## 3.3 ENUMERĂRI

### 3.3.1 Colors

Conține 3 culori care pot fi folosite atât pentru tabla de șah, cât și pentru piese: **WHITE**, **BLACK**, **GRAY** (o puteți folosi ca pe o culoare ajutătoare, cum ar fi pentru marginile tablei de șah, sau pentru pătrățele ocupate, etc.).

# 4 Excepții

Pentru ca jocul să funcționeze corect în toate cazurile, va trebui să tratați anumite excepții care pot apărea în timpul jocului. Astfel, se vor arunca următoarele excepții, cu mesaje sugestive:

- `InvalidCommandException` - dacă utilizatorul introduce o comandă invalidă (de exemplu, un format greșit pentru mutare, o opțiune inexistentă în meniu, text acolo unde se așteaptă un număr etc.).
- `InvalidMoveException` - dacă se încearcă mutarea interzisă a unei piese, de exemplu: coordonate în afara tablei, încălcarea regulilor de deplasare pentru tipul de piesă, sărit peste alte piese atunci când nu este permis, mutarea unei piese care nu aparține jucătorului curent sau o mutare care lasă propriul rege în șah. În mod tipic, această excepție va fi aruncată din metode precum `Board.isValidMove` sau `Player.makeMove`.

**Atenție!**

Veți pierde puncte dacă în timpul utilizării jocului apar excepții pe care nu le tratați.

Sunteți liberi să adăugați orice alte tipuri de excepții.

În practică, un utilizator poate folosi greșit jocul (având în vedere că se bazează pe un input al utilizatorului) și se poate ajunge la un comportament imprevizibil sau erori ale jocului. De exemplu, utilizatorul poate introduce un șir de caractere acolo unde se așteaptă un număr întreg. Mai departe, în aplicație, acel șir de caractere este folosit într-o ecuație matematică, ceea ce ar produce o eroare. Trebuie să vă asigurați că nu apar astfel de cazuri în timpul jocului **(programare defensivă)**.

## 5 Flow-ul jocului

Jocul trebuie să urmărească pașii de mai jos (acești pași vor fi orchestrați, în principal, de metoda `Main.run()`, care va apela metode din clasele `User` și `Game`):

1. Utilizatorul trebuie să completeze credențialele pentru a se autentifica în cont. Dacă autentificarea eșuează (credențialele nu corespund), va fi pus să introducă alte credențiale până când datele sunt introduse corect. De asemenea, aplicația trebuie să ofere posibilitatea creării unui cont nou (apelând metoda `newAccount`).
2. După autentificare se vor afișa opțiunile pe care le are utilizatorul:
   - Începerea unui joc nou:
     - Player vs. Computer
       * Se va introduce alias pentru Player și se va alege o culoare. La inițierea jocului, se vor crea obiectele `Game` și `Player`, iar toate punctele acumulate pe parcursul jocului de Player vor fi salvate, la sfârșitul jocului, în contul utilizatorului curent.
       * Se apelează metoda `Game.start()` și începe jocul de șah.
   - Vizualizare jocuri în progres:
     - Se va afișa o listă cu jocurile începute de utilizator (corespunzătoare obiectelor `Game` asociate `User`-ului curent).
     - Utilizatorul poate să selecteze un joc (folosind identificatorul jocului) și să:
       * vizualizeze toate detaliile despre acesta (jucătorii, reprezentare acuală a tablei, istoricul mutărilor, etc..); după afișarea acestora se va reveni la meniul principal);
       * continue jocul de la stadiul curent (se apelează metoda dedicată reluării jocului, `Game.resume()`, care porneşte de la starea salvată);
       * șterge jocul din listă (jocul este eliminat din colecția de jocuri a utilizatorului; după ștergere se revine la meniul principal).
   - Delogare: dacă utilizatorul alege această opțiune, aplicația îi oferă posibilitatea fie să se autentifice din nou, fie să închidă complet aplicația.
3. Aplicația **trebuie să execute** opțiunea aleasă de utilizator, așa cum este descris în cerință, prin apeluri corespunzătoare către metodele claselor `Main`, `User` și `Game`.

### 5.1 Desfășurarea jocului

Jocul (o instanță a clasei `Game`, cu tabla de tip `Board` și jucătorii de tip `Player`) se desfășoară în felul următor:

1. Se afișează tabla de șah, din perspectiva utilizatorului (nu a computerului). Un exemplu de afișare este reprezentat în Figura 2. Piesele sunt reprezentate de un caracter, așa cum a fost specificat anterior - K (king), Q (queen), etc.. Fiecare caracter este urmat de litera ce reprezintă culoarea piesei: K-W - regele alb, K-B - regele negru. **Puteți afișa în orice mod tabla de șah, cât timp se pot diferenția piesele după rolul și culoarea lor.**
2. Utilizatorul are 4 variante:
   - Introduce de la tastatură poziția uneia dintre piesele pe care le deține - de exemplu "B2". Dacă piesa de la acea poziție îi aparține, se vor afișa pe ecran mutările posibile pe care le poate face, folosind metoda `getPossibleMoves` a piesei și informațiile din `Board` (pe exemplul din Figura 2 se vor afișa B3, B4).
   - Introduce de la tastatură mutarea pe care dorește să o facă, sub forma "poziție_actuală-poziție_nouă". Pentru exemplul ales, se mută pionul de la B2 un pătrat înainte -

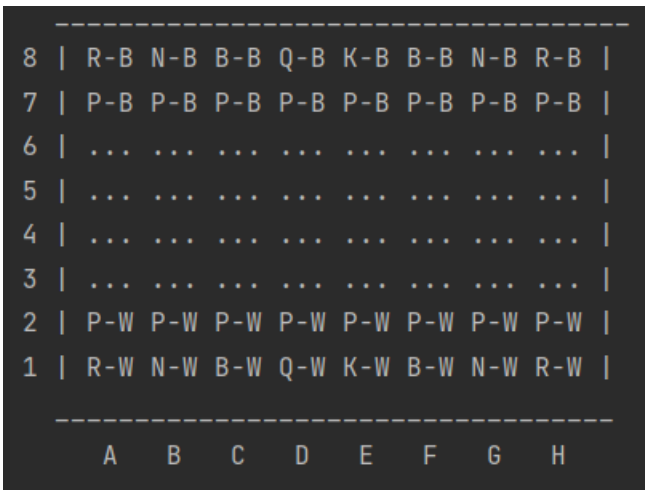


Figura 2: Exemplu de afișare a tablei de șah pentru un joc nou

"B2-B3". Rezultatul poate fi văzut în Figura 3. Fiecare mutare făcută de orice jucător va fi salvată într-un istoric al mutărilor, de exemplu prin metoda `Game.addMove`.

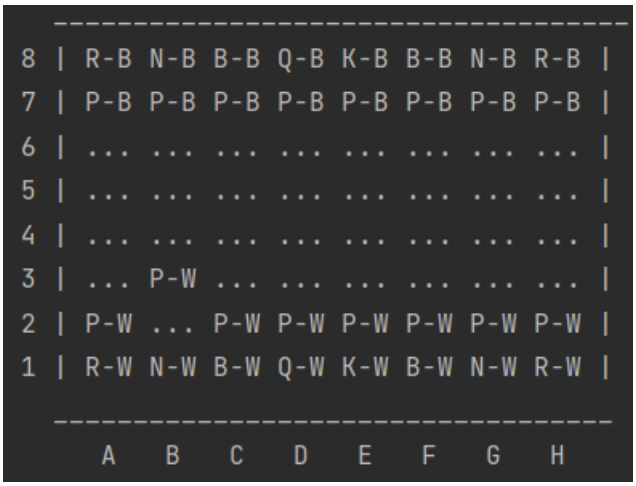


Figura 3: Mutarea pionului

- Renunță, acceptându-și înfrângerea. În acest caz, la punctele pe care le deține jucătorul, se vor adăuga punctele acumulate în joc și se vor scădea 150 de puncte pentru înfrângere, conform regulilor de punctaj. Jocul este marcat ca încheiat.
- Părăsește jocul. În acest caz, starea curentă a obiectului `Game` (tabla, mutările, punctajul curent etc.) se salvează în lista de jocuri în desfășurare a utilizatorului și poate fi continuată la cerere.

3. După ce jucătorul a făcut o mutare validă (verificată prin `Board.isValidMove` / `Player.makeMove`), **calculatorul va alege o piesă random din cele disponibile și o mutare random din cele posibile**, pe baza listei de mutări generate de metoda `getPossibleMoves` pentru piesele controlate de computer. După mutarea computerului, se schimbă jucătorul curent prin `Game.switchPlayer()` și jocul continuă.

Observații

- Toate acțiunile din joc trebuie să fie însoțite de mesaje explicative (de exemplu "Next move: Computer", "Invalid command", "Invalid move", "You are in check" etc.).
- Asigurați-vă că tratați **TOATE** excepțiile care pot apărea, folosind clase precum `InvalidCommandException` și `InvalidMoveException`. De exemplu, introducerea unui input greșit pentru o mutare "J9-K3" trebuie să fie interceptată și să ducă la un mesaj clar, fără oprirea neașteptată a programului.

## 5.2 Finalul jocului

Jocul se poate termina atunci când:

- Jucătorul părăsește jocul. În acest caz, jocul este adăugat în lista de jocuri aflate în desfășurare în contul său (lista de `Game` din `User`), putând fi reluat ulterior.
- Jucătorul renunță, jocul fiind câștigat automat de competitor. **Considerăm că oponentul Computer renunță atunci când se ajunge într-un caz de egalitate**. Considerăm caz de egalitate în situația în care și computerul și utilizatorul pleacă și revin în aceeași poziție a unei piese de 3 ori consecutiv (detecția acestui caz poate fi realizată în clasa `Game`, folosind istoricul mutărilor).
- Unul dintre jucători reușește să aducă în șah-mat regele adversarului. În acest caz, jocul este marcat ca finalizat, se aplică regulile de punctaj, iar informațiile relevante se salvează în obiectele `User` și, dacă este cazul, în fișierele de intrare/ieșire.

## 5.3 Puncte acumulate în joc

Utilizatorul (presupunem că Player1 este utilizatorul autentificat și Player2 computerul) deține în cont $X$ puncte (0 la crearea unui utilizator nou). Acestea se modifică la finalizarea fiecărui joc, după cum urmează. Notăm cu $Y$ totalul punctelor acumulate în timpul jocului curent (din piesele capturate, conform tabelului de mai jos).

- Dacă Player2 renunță, jocul se termină, iar la punctele actuale $X$ ale lui Player1 se adaugă punctele acumulate în timpul jocului $Y$ și se adaugă 150 de puncte pentru câștigarea jocului. Dacă Player1 renunță, se scad 150 de puncte pentru părăsirea jocului.

$$X_{\text{nou}} = X + Y \pm 150$$

(se adună +150 la victorie, se scad −150 la renunțare).

- Dacă Player1 aduce în șah-mat regele adversarului, jocul se termină, iar la punctele actuale $X$ ale lui Player1 se adaugă punctele acumulate în timpul jocului $Y$ și se adaugă 300 de puncte pentru câștigarea jocului. Dacă Player2 aduce în șah-mat regele lui Player1, se scad 300 de puncte pentru pierderea jocului.

$$X_{\text{nou}} = X + Y \pm 300$$

(se adună +300 la victorie prin șah-mat, se scad −300 la înfrângere).

Pentru fiecare piesă capturată, se vor adăuga la punctaj sumele reprezentate în tabel:

| Piesa | Puncte |
|---|---|
| Queen | 90 |
| Rook | 50 |
| Bishop | 30 |
| Knight | 30 |
| Pawn | 10 |

## 6 Fișiere de intrare

Pentru testarea aplicației voastre, veți folosi fișierele JSON puse la dispoziție.

- **accounts.json** – detaliile despre toți utilizatorii creați (email, parolă, puncte totale, lista de identificatori ai jocurilor aflate în derulare).
  De exemplu:

```
[
  {
    "email": "player@example.com",
    "password": "parola123",
    "points": 120,
    "games": [1, 3]
  }
]
```

- **games.json** – detaliile despre toate jocurile începute și nefinalizate (identificatorul computerul are email-ul "computer", jucătorii, starea curentă a tablei, rândul curent și istoricul mutărilor care conține și piesele capturate).
  De exemplu, pentru un joc nou (tabla în poziția inițială, fără mutări efectuate):

```
[
  {
    "id": 1,
    "players": [
      {
        "email": "ana@example.com",
        "color": "WHITE"
      },
      {
        "email": "computer",
        "color": "BLACK"
      }
    ],
    "currentPlayerColor": "WHITE",
    "board": [
      {
        "type": "R",
        "color": "WHITE",
        "position": "A1"
      },
      {
        "type": "N",
        "color": "WHITE",
        "position": "B1"
      },
      {
        "type": "P",
        "color": "WHITE",
```

```
        "position": "A2"
      }
      /* ... restul pieselor inițiale ... */
    ],
    "moves": []
  }
]
```

În fișierul real, în câmpul `"board"` vor fi prezente toate piesele de pe tablă, fiecare cu tipul, culoarea și poziția ei curentă. Lista `"moves"` va conține mutările efectuate și piesele capturate la fiecare mutare, ca în exemplu.

```
"moves": [
      {
        "playerColor": "WHITE",
        "from": "E2",
        "to": "E4"
      },
      {
        "playerColor": "BLACK",
        "from": "E7",
        "to": "E5"
      },
      {
        "playerColor": "WHITE",
        "from": "G1",
        "to": "F3"
      },
      {
        "playerColor": "BLACK",
        "from": "B8",
        "to": "C6"
      },
      {
        "playerColor": "WHITE",
        "from": "F3",
        "to": "E5",
        "captured": {
          "type": "P",
          "color": "BLACK"
        }
      }
    ]
```

Fiecare joc are asociat un ID. În lista de jocuri ale utilizatorului (fișierul *accounts.json*), se vor afla ID-urile care ajută la identificarea jocurilor corespunzătoare din fișierul *games.json*.

Pentru manipularea datelor în format JSON, puteți folosi orice bibliotecă de parsare JSON. Recomandăm, cu titlu de exemplu, biblioteca `json-simple`, care poate fi descărcată accesând acest **link**.

În plus, în arhiva temei este disponibilă o clasă care se ocupă de parsarea corectă a fișierelor de intrare în format JSON. Puteți utiliza direct această clasă sau vă puteți defini propria implementare, atâta timp cât respectați formatul de intrare și ieșire specificat în enunț.

## 7 Observații

Atenție!

- **Tipizați** orice colecție folosită în cadrul implementării.
- Respectați specificațiile detaliate în enunțul temei și folosiți indicațiile menționate.
- Puteți adăuga orice alte detalii și funcționalități pentru joc.
- Puteți aduce modificări asupra fișierelor de input (de exemplu, pentru adăugarea unor flaguri care v-ar fi utile), dar fără a modifica detaliile deja existente.
- În programarea orientată pe obiecte, **încapsularea** este un principiu fundamental, folosit pentru ascunderea detaliilor și protecția datelor. Deși este considerată o practică profesionistă în dezvoltarea software-ului, nu impunem ca toate clasele arhitecturii să fie implementate folosind acest principiu.
- Pentru orice fel de întrebări, puteți folosi forumul.
- **Toate elementele care nu sunt specificate în cerință rămân la alegerea voastră.**

## 8 Testare

Testarea jocului se va face în timpul prezentării temei de la laborator. **Veți primi punctaj doar pe funcționalitățile care pot fi testate în timpul prezentării.**

Atenție!

- Fiecare componentă creată trebuie să fie testată.
- **Nu se va puncta ierarhia de clase, fără ca acestea să fie testate**.
- **Va trebui să vă creați una sau mai multe metode main care testează toate funcționalitățile claselor implementate, altfel nu veți primi punctajul!**
- Asistentul de la laborator poate solicita explicarea anumitor secvențe de cod, dacă el consideră necesar acest lucru.

## 9 Notare

| Cerința | Puncte |
|---|---|
| Implementarea integrală a arhitecturii și funcționalităților propuse<br>și testarea acestora | 0.7 |

Atenție!

- Tema (prima parte) valorează **0.7 puncte** din nota finală a disciplinei POO!
- **Partea a II-a a temei este dependentă de această primă parte. Nu veți primi punctaj pe partea a II-a a temei dacă prima parte nu este implementată.**
- Tema este **individuală**! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.
- **Se vor puncta DOAR funcționalitățile care pot fi testate.** Acest lucru înseamnă că va trebui să aveți o clasă de test prin care **să oferiți** posibilitatea **testării** tuturor funcționalităților pe care le-ați implementat, **altfel nu veți primi punctaj**.
- Tema se va încărca pe site-ul de cursuri până la termenul specificat în pagina de titlu. Se va trimite o arhivă **.zip** ce va avea un nume de forma **grupa_Nume_Prenume_tema1.zip** (ex. 326CC_Popescu_Andreea_tema1.zip) și care va conține următoarele:
  - un folder **SURSE** ce conține doar sursele Java;
  - un folder **PROIECT** ce conține proiectul în mediul ales de voi (de exemplu NetBeans, Eclipse, Intellij IDEA);
  - un fișier **README.pdf** în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți specifica, pe scurt, în ce a constat dificultatea implementării.
- Lipsa fișierului README sau nerespectarea formatului impus pentru arhivă duce la o **depunctare** de **0.1** (fiecare) din punctajul temei.