Medii și Instrumente de Programare

Utilizarea Laboratoarelor de JAVA în Proiectul VoteSystem

Bondoc Daniel

Introducere

Proiectul nostru, *VoteSystem*, a fost dezvoltat utilizând conceptele învățate pe parcursul laboratoarelor de programare Java. Fiecare laborator a contribuit semnificativ la dezvoltarea aplicației, aceasta incluzând funcționalități care au la bază conceptele detaliate în laboratoare.

Aplicația *VoteSystem* își propune să trateze un sistem simplu de votare care se aplică pentru candidați explicit menționați și pentru sondaje de tip poll care numără voturile pentru fiecare opțiune numerică specificată.

Programul include mai multe funcționalități printe care se numără votul pentru un candidat, votul pentru o opțiune de sondaj, afișarea statisticilor de la momentul actual, salvarea voturilor într-un fișier JSON si citirea acestora dintr-un fișier similar.

Această documentație detaliază modul în care fiecare tematică a fost aplicată, incluzând exemple relevante de cod.

Laborator 1: Noțiuni de bază Java

În laboratorul 1, am învățat despre tipuri de date și casting. Aceste concepte au fost aplicate în *VoteSystem* pentru stocarea voturilor. De exemplu, variabilele de tip `int` sunt folosite pentru a reprezenta numărul de voturi, iar castingul este utilizat pentru validarea inputului utilizatorului.

```
    Scanner scanner = new Scanner(System.in);
    System.out.print("Introduceti optiunea: ");
    int choice = 0;
    try {
```

```
5         choice = Integer.parseInt(scanner.nextLine());
6       } catch (NumberFormatException e) {
7         System.out.println("Input invalid. Introduceti un numar
      .");
8       }
9
```

Listing 1: Exemplu de casting pentru validarea inputului

Acest laborator ne-a ajutat să înțelegem cum să gestionăm și să validăm datele primite de la utilizator.

Se execută citirea sigură a unui număr întreg introdus de utilizator, folosind un obiect Scanner. Utilizăm metoda scanner.nextLine() pentru a prelua inputul ca șir, iar apoi încercăm să-l convertim în număr folosind Integer.parseInt(). În cazul unei valori invalide (de exemplu, litere sau simboluri), este generată o excepție NumberFormatException, care este gestionată în blocul catch, afișând un mesaj de eroare. Astfel, programul evită blocarea și oferă feedback utilizatorului pentru a introduce un număr valid. Această abordare este utilă pentru prevenirea erorilor și îmbunătățirea interacțiunii utilizatorului cu aplicația.

Laborator 2: Funcții, metode, tipuri returnate

În laboratorul 2, ne-au fost explicate funcțiile și metodele din JAVA. Acestea au fost fundamentale pentru modularizarea codului în *VoteSystem*. De exemplu, metoda `castVote()` din clasa `VoteCandidate` gestionează logica unui vot individual.

```
1     @Override
2     public void castVote(String candidate) {
3       votes.put(candidate, votes.getOrDefault(candidate, 0) +
      1);
4       System.out.println("Vot adaugat pentru " + candidate);
5     }
6
```

Listing 2: Exemplu de metodă pentru procesarea voturilor

Această metodă simplifică procesul de actualizare a voturilor și evidențiază importanța separării logicii în funcții.

Această metodă suprascrie comportamentul din clasa părinte *Vote*, folosind adnotarea @Override. Metoda adaugă un vot pentru un candidat în harta votes, utilizând votes.getOrDefault() pentru a obține numărul curent de voturi (sau 0 dacă nu are voturi). Incrementăm voturile și actualizăm harta cu noua valoare. Apoi, afișăm un mesaj pentru utilizator, confirmând că votul a fost adăugat.

Laborator 3: Clase și moștenire

Conceptul de clase și moștenire a fost folosit pentru a crea o ierarhie logică eficientă în proiect. De exemplu, `VoteCandidate` și `VotePoll` moștenesc din clasa de bază `Vote`, care conține logica comună pentru gestionarea voturilor.

```
    public abstract class Vote implements VotingSystem {
      protected Map<String, Integer> votes = new HashMap<>();
      @Override
      public void castVote(String option) {
        votes.put(option, votes.getOrDefault(option, 0) + 1);
      }

      @Override
      public Map<String, Integer> getVotes() {
        return votes;
      }

      public abstract void specificVoteBehavior(String input)
;
    }

    public class VoteCandidate extends Vote {
      @Override
      public void specificVoteBehavior(String candidate) {
        System.out.println("Casting vote for candidate: " +
candidate);
        castVote(candidate);
      }
    }

    public class VotePoll extends Vote {
      @Override
      public void specificVoteBehavior(String pollOption) {
        System.out.println("Casting vote for poll option: " +
    pollOption);
        castVote(pollOption);
      }
    }

```

Listing 3: Exemplu de moștenire

Moștenirea a permis implementarea logicii generale de gestionare a voturilor în clasa de bază `Vote`, reducând duplicarea codului în clasele derivate. Clasele `VoteCandidate` și `VotePoll` extind această funcționalitate pentru tipuri specifice de vot, demonstrând beneficiile polimorfismului și reutilizării codului.

Am folosit clase și funcții abstracte pentru a centraliza logica comună,

precum gestionarea voturilor, în clasa abstractă `Vote`. Aceasta implementează metodele comune, iar metoda abstractă `specificVoteBehavior` obligă clasele derivate să definească comportamente specifice (ex. voturi pentru candidați sau sondaje). Interfața `VotingSystem` asigură implementarea unor metode standard pentru toate tipurile de voturi. Acest design îmbunătățește reutilizarea codului și extinderea aplicației.

Laborator 4: Interfețe

Interfețele au fost utile pentru definirea unui contract comun între diferitele tipuri de voturi. De exemplu, `VotingSystem` implementează interfața `IVoteStatistics`, care definește metode pentru obținerea statisticilor voturilor.

```
    public interface VotingSystem {
      void castVote(String option);
      Map<String, Integer> getVotes();
    }
  }

```

Listing 4: Exemplu de interfață

Interfața VotingSystem definește două metode esențiale pentru un sistem de vot. Metoda `castVote(String option)` este utilizată pentru a înregistra un vot pentru o opțiune specificată, iar metoda `getVotes()` returnează harta cu opțiunile și numărul de voturi asociate fiecărei opțiuni.

Acest lucru asigură că fiecare tip de vot implementează logica statisticilor într-un mod specific.

Laborator 5: Proiectarea aplicației

În laboratorul 5, am învățat despre cerințele de proiectare. Acest lucru a fost aplicat pentru a crea arhitectura proiectului, incluzând clase separate pentru salvarea datelor, statistici și logica principală. Design-ul orientat pe responsabilități a facilitat integrarea tuturor funcționalităților dorite.

Grafic UML pentru VoteSystem

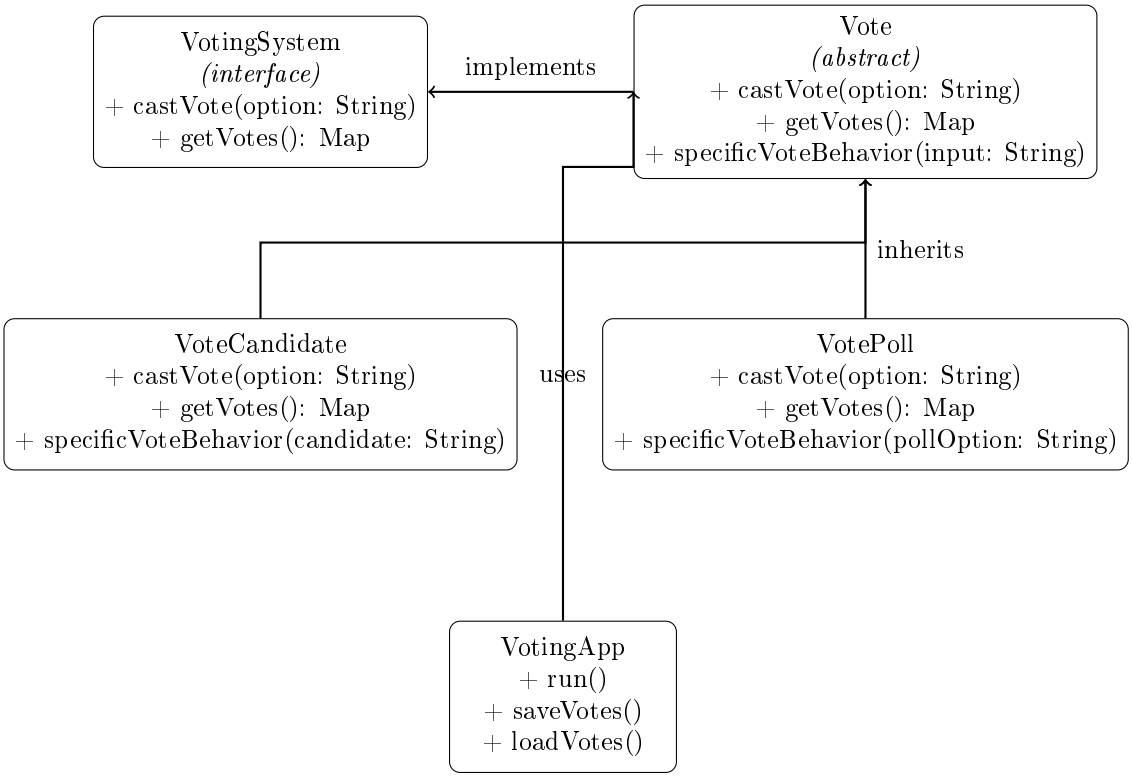
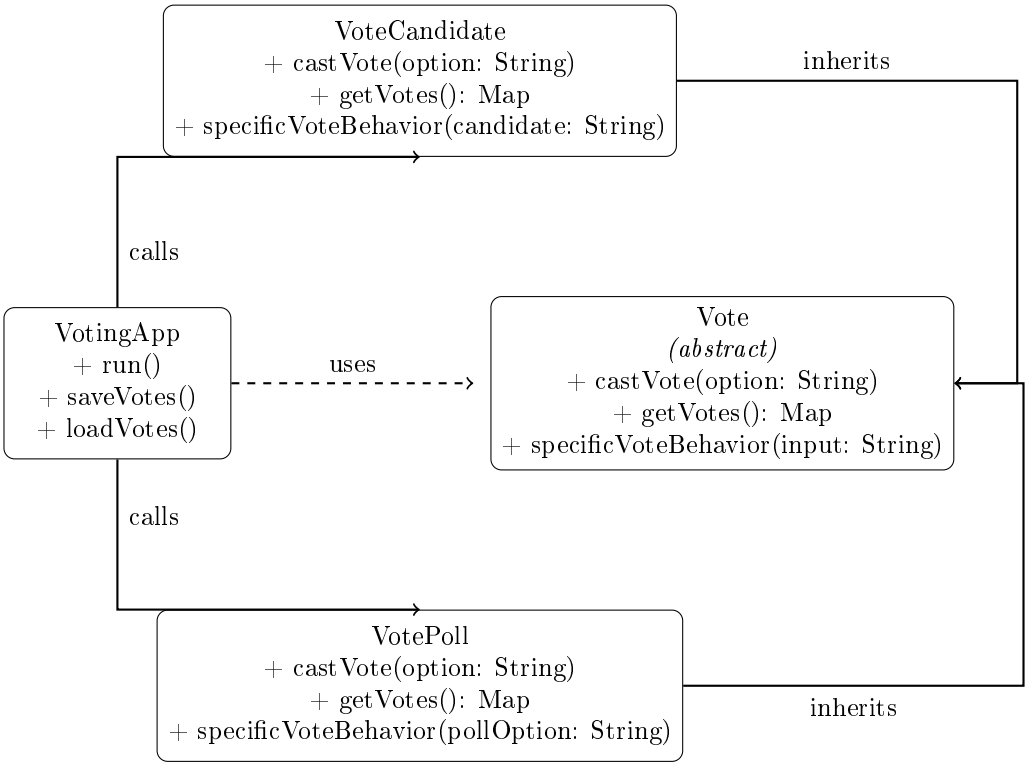
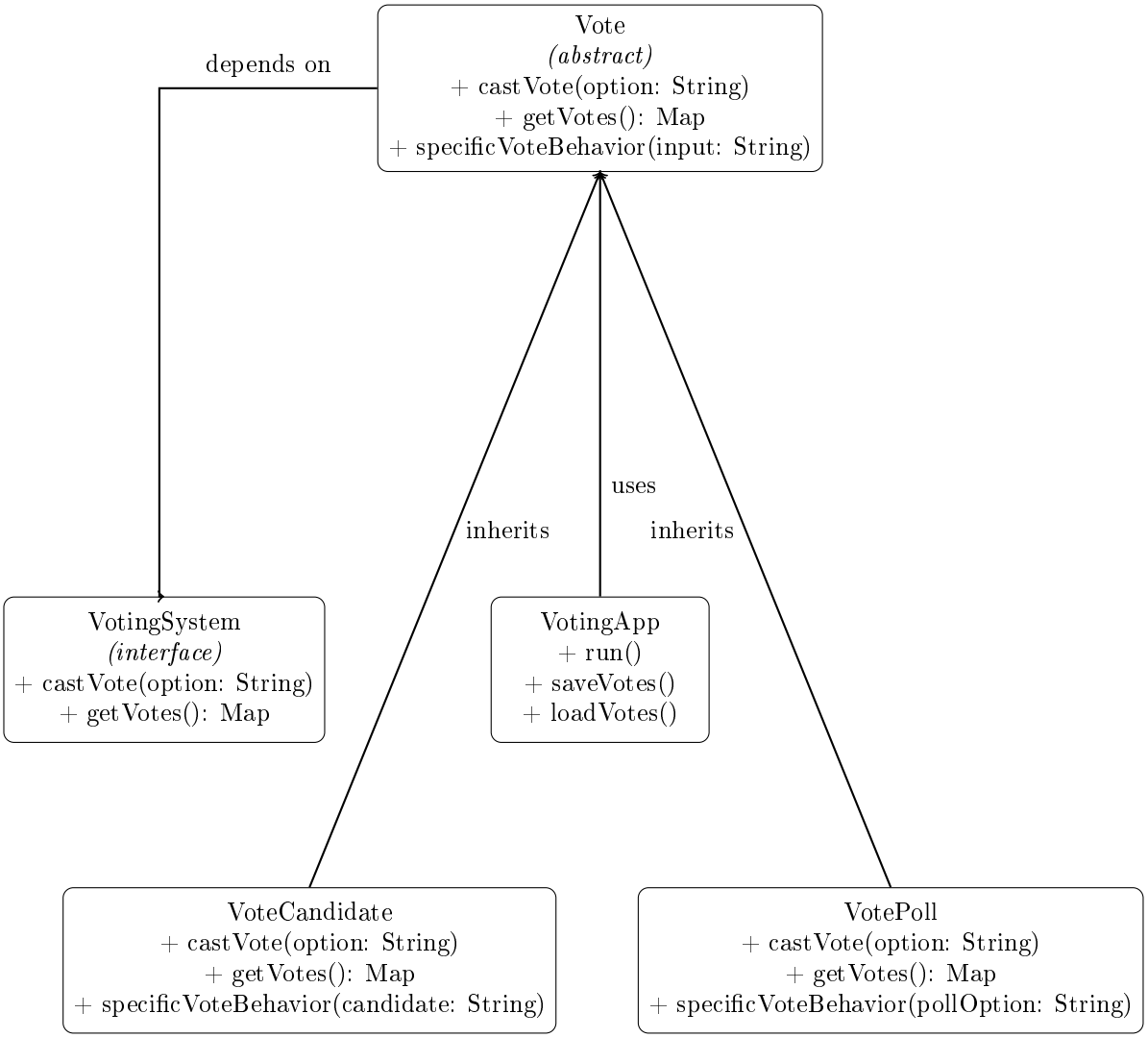


Diagrama de Interacțiune dintre Clasele din VoteSystem



Rolul și interdependența claselor din VoteSystem



Laborator 6: Fișiere JSON

În *VoteSystem*, fișierele JSON sunt utilizate pentru salvarea și încărcarea voturilor și candidaților. Am folosit biblioteca Jackson pentru serializare.

```
    private static final ObjectMapper mapper = new
ObjectMapper();
    public static void saveVotesToJson(Vote vote, String
filePath) {
       try {
         mapper.writeValue(new File(filePath), vote.getVotes()
);
         System.out.println("Votes have been saved to " +
filePath);
       } catch (IOException e) {
         e.printStackTrace();
       }
    }

```

Listing 5: Exemplu de salvare în JSON

Laborator 7: Testare unitară

Testarea unitară este fundamentală pentru validarea funcționalității aplicației. De exemplu, testele pentru înregistrarea voturilor verifică corectitudinea metodei `castVote()`.

```
    @Test
    public void testRecordVote() {
      VoteCandidate candidateVote = new VoteCandidate();
      candidateVote.castVote("Alice");
      candidateVote.castVote("Alice");
      candidateVote.castVote("Bob");

      Map<String, Integer> votes = candidateVote.getVotes();
      assertEquals(2, (int)votes.get("Alice"));
      assertEquals(1, (int)votes.get("Bob"));
    }

```

Listing 6: Exemplu de test unitar

Casting-ul `(int)votes.get("Alice")` este folosit pentru a transforma un obiect `Integer` într-un tip primitiv int, necesar pentru comparația din `assertEquals`. Astfel este asigur că tipul valorii extrase din mapă corespunde cu tipul primitiv așteptat în test.