```
import java.util.*;

// Factory for creating core entities (Voter, Candidate)
interface VotingFactory {
   Voter createVoter(String name, String password);
   Candidate createCandidate(String name, String party);
}

// Concrete Factory for the Online Voting System
 class VotingSystemFactory implements VotingFactory {
   @Override
   public Voter createVoter(String name, String password) {
      return new Voter(name, password);
   }

   @Override
   public Candidate createCandidate(String name, String party) {
      return new Candidate(name, party);
   }
}

// Core classes for Voter, Candidate, and other entities
 class Voter {
   private String name;
   private String password;
   private boolean hasVoted = false;

   public Voter(String name, String password) {
        this.name = name;
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public String getPassword() {
        return password;
    }

    public boolean hasVoted() {
        return hasVoted;
    }

    public void vote() {
        this.hasVoted = true;
    }
}

 class Candidate {
    private String name;
    private String party;
    private String encryptedVotes;

    public Candidate(String name, String party) {
        this.name = name;
        this.party = party;
    }

    public String getName() {
        return name;
    }

    public String getParty() {
        return party;
    }

    public String getEncryptedVotes() {
        return encryptedVotes;
    }

    public void setEncryptedVotes(String encryptedVotes) {
        this.encryptedVotes = encryptedVotes;
    }
}

// Authentication Strategy interface and concrete implementation
interface AuthenticationStrategy {
    boolean authenticate(Voter voter, VoterDatabase voterDatabase);
}

 class PasswordAuthenticationStrategy implements AuthenticationStrategy {
    @Override
    public boolean authenticate(Voter voter, VoterDatabase voterDatabase) {
        Voter storedVoter = voterDatabase.getVoter(voter.getName());
        if (storedVoter != null && storedVoter.getPassword().equals(voter.getPassword()) &&
!storedVoter.hasVoted()) {
            storedVoter.vote();
            return true;
        }
        return false;
    }
}

// Encryption Strategy interface and concrete implementation
interface EncryptionStrategy {
    String encrypt(int value);
}

 class BasicEncryptor implements EncryptionStrategy {
    @Override
    public String encrypt(int value) {
        // Implement encryption logic here
        return "encrypted_" + value;
    }
}

// Observer interface for vote notifications
interface Observer {
    void update(String message);
}
```

```
}

// Ballot observer to notify when a vote is cast
 class BallotObserver implements Observer {
    @Override
    public void update(String message) {
        System.out.println("Ballot Update: " + message);
    }
}

// Ballot class using Observer pattern for notifications
 class Ballot {
    private Map<Candidate, Integer> votes;
    private List<Observer> observers;

    public Ballot() {
        this.votes = new HashMap<>();
        this.observers = new ArrayList<>();
    }

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    private void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
    public void addVote(Candidate candidate) {
        votes.put(candidate, votes.getOrDefault(candidate, 0) + 1);
        notifyObservers("Vote added for candidate: " + candidate.getName());
    }

    public int getVotes(Candidate candidate) {
        return votes.getOrDefault(candidate, 0);
    }
}

// Command pattern for casting votes
interface Command {
    void execute();
}

// Command to cast a vote
 class CastVoteCommand implements Command {
    private Voter voter;
    private Candidate candidate;
    private Ballot ballot;
    private AuthenticationStrategy authenticationStrategy;
    private VoterDatabase voterDatabase;

    public CastVoteCommand(Voter voter, Candidate candidate, Ballot ballot,
AuthenticationStrategy authStrategy, VoterDatabase voterDatabase) {
        this.voter = voter;
        this.candidate = candidate;
        this.ballot = ballot;
        this.authenticationStrategy = authStrategy;
        this.voterDatabase = voterDatabase;
    }

    @Override
    public void execute() {
        if (authenticationStrategy.authenticate(voter, voterDatabase)) {
            ballot.addVote(candidate);
        } else {
            System.out.println("Authentication failed or voter has already voted.");
        }
    }
}

// OnlineVotingSystem class using design patterns
 class OnlineVotingSystem {
    private VoterDatabase voterDatabase;
    private CandidateDatabase candidateDatabase;
    private Ballot ballot;
    private EncryptionStrategy encryptionStrategy;
    private AuthenticationStrategy authenticationStrategy;

    public OnlineVotingSystem(VotingFactory factory) {
        this.voterDatabase = new VoterDatabase();
        this.candidateDatabase = new CandidateDatabase();
        this.ballot = new Ballot();
        this.encryptionStrategy = new BasicEncryptor(); // Can be swapped for a different
encryption strategy
```

```
        this.authenticationStrategy = new PasswordAuthenticationStrategy(); // Can be
swapped for a different auth strategy
    }


    public void registerVoter(String name, String password) {
        Voter voter = new Voter(name, password);
        voterDatabase.addVoter(voter);
    }


    public void addCandidate(String name, String party) {
        Candidate candidate = new Candidate(name, party);
        candidateDatabase.addCandidate(candidate);
    }
public Voter getVoter(String name) {
    return voterDatabase.getVoter(name);
}


public Candidate getCandidate(String name) {
    for (Candidate candidate : candidateDatabase.getCandidates()) {
        if (candidate.getName().equals(name)) {
            return candidate;
        }
    }
    return null; // Return null if no candidate is found
}


    public void castVote(Voter voter, Candidate candidate) {
        Command castVote = new CastVoteCommand(voter, candidate, ballot,
authenticationStrategy, voterDatabase);
```

```
        castVote.execute();
    }

    public List<Candidate> getResults() {
        List<Candidate> results = new ArrayList<>();
        for (Candidate candidate : candidateDatabase.getCandidates()) {
            int votes = ballot.getVotes(candidate);
            String encryptedVotes = encryptionStrategy.encrypt(votes);
            candidate.setEncryptedVotes(encryptedVotes);
            results.add(candidate);
        }
        return results;
    }
}

// VoterDatabase managing voter records
 class VoterDatabase {
    private final List<Voter> voters;

    public VoterDatabase() {
        this.voters = new ArrayList<>();
    }

    public void addVoter(Voter voter) {
        voters.add(voter);
    }

    public Voter getVoter(String name) {
        for (Voter voter : voters) {
            if (voter.getName().equals(name)) {
                return voter;
            }
        }
        return null;
    }
}

// CandidateDatabase managing candidate records
 class CandidateDatabase {
    private final List<Candidate> candidates;

    public CandidateDatabase() {
        this.candidates = new ArrayList<>();
    }

    public void addCandidate(Candidate candidate) {
        candidates.add(candidate);
    }

    public List<Candidate> getCandidates() {
        return candidates;
    }
}

// Main class demonstrating usage
public class Abishek {
```

```java
public static void main(String[] args) {
    VotingFactory factory = new VotingSystemFactory();
    OnlineVotingSystem votingSystem = new OnlineVotingSystem(factory);

    // Register voters (should be done before fetching them)
    votingSystem.registerVoter("Alice", "password1");
    votingSystem.registerVoter("Bob", "password2");

    // Add candidates
    votingSystem.addCandidate("John Doe", "Party A");
    votingSystem.addCandidate("Jane Smith", "Party B");

    // Fetch voter and candidate from the database
    Voter alice = votingSystem.getVoter("Alice");
    Candidate johnDoe = votingSystem.getCandidate("John Doe");
        Voter bob = votingSystem.getVoter("Bob");
    Candidate janesmith = votingSystem.getCandidate("Jane Smith");



    if (alice != null && johnDoe != null) {
        votingSystem.castVote(alice, johnDoe);
    } else {
        System.out.println("Invalid voter or candidate.");
    }
        if (bob != null && janesmith != null) {
        votingSystem.castVote(bob, janesmith);
    } else {
        System.out.println("Invalid voter or candidate.");
        }



        // Get and display election results

        List<Candidate> results = votingSystem.getResults();

        for (Candidate candidate : results) {

            System.out.println("Candidate: " + candidate.getName() + ", Encrypted Votes: " +
candidate.getEncryptedVotes());

        }

    }

}
```