```
import java.io.*;
import java.security.*;
import java.util.Base64;
import java.util.Date;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import java.util.Scanner;

// Custom exception for banking operations
class BankingException extends Exception {
    public BankingException(String message) {
        super(message);
    }
}

// Customer class implementing Serializable
class Customer implements Serializable {
    private static final long serialVersionUID = 1L;

    // Regular customer information
    private String customerId;
    private String fullName;
    private String address;
    private String phoneNumber;
    private Date accountCreationDate;
    private double accountBalance;

    // Sensitive information - marked as transient to exclude from default serialization
    private transient String socialSecurityNumber;
    private transient String password;
    private transient String creditCardNumber;

    public Customer(String customerId, String fullName, String address,
                    String phoneNumber, String socialSecurityNumber,
                    String password, String creditCardNumber, double initialBalance) {
        this.customerId = customerId;
        this.fullName = fullName;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.socialSecurityNumber = socialSecurityNumber;
        this.password = password;
        this.creditCardNumber = creditCardNumber;
        this.accountBalance = initialBalance;
        this.accountCreationDate = new Date();
    }

    // Custom serialization method
    private void writeObject(ObjectOutputStream oos) throws IOException {
        oos.defaultWriteObject(); // Serialize non-transient fields

        // Encrypt sensitive fields before serialization
        try {
            String encryptedSSN = EncryptionUtil.encrypt(socialSecurityNumber);
            String encryptedPassword = EncryptionUtil.encrypt(password);
            String encryptedCreditCard = EncryptionUtil.encrypt(creditCardNumber);

            oos.writeUTF(encryptedSSN);
            oos.writeUTF(encryptedPassword);
            oos.writeUTF(encryptedCreditCard);
        } catch (Exception e) {
            throw new IOException("Error encrypting sensitive data during serialization", e);
```

```
    }
}

// Custom deserialization method
private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    ois.defaultReadObject(); // Deserialize non-transient fields

    // Decrypt sensitive fields after deserialization
    try {
        String encryptedSSN = ois.readUTF();
        String encryptedPassword = ois.readUTF();
        String encryptedCreditCard = ois.readUTF();

        this.socialSecurityNumber = EncryptionUtil.decrypt(encryptedSSN);
        this.password = EncryptionUtil.decrypt(encryptedPassword);
        this.creditCardNumber = EncryptionUtil.decrypt(encryptedCreditCard);
    } catch (Exception e) {
        throw new IOException("Error decrypting sensitive data during deserialization", e);
    }
}

// Getters and setters
public String getCustomerId() { return customerId; }
public String getFullName() { return fullName; }
public String getAddress() { return address; }
public String getPhoneNumber() { return phoneNumber; }
public double getAccountBalance() { return accountBalance; }
public Date getAccountCreationDate() { return accountCreationDate; }

// Sensitive data getters - only return masked versions for security
public String getMaskedSSN() {
    return "XXX-XX-" + socialSecurityNumber.substring(7);
}

public String getMaskedCreditCard() {
    return "XXXX-XXXX-XXXX-" + creditCardNumber.substring(12);
}

public boolean verifyPassword(String inputPassword) {
    return this.password.equals(inputPassword);
}

// Banking operations
public void deposit(double amount) throws BankingException {
    if (amount <= 0) {
        throw new BankingException("Deposit amount must be positive");
    }
    this.accountBalance += amount;
}

public void withdraw(double amount) throws BankingException {
    if (amount <= 0) {
        throw new BankingException("Withdrawal amount must be positive");
    }
    if (amount > accountBalance) {
        throw new BankingException("Insufficient funds");
    }
    this.accountBalance -= amount;
}

public void transfer(Customer recipient, double amount) throws BankingException {
    if (amount <= 0) {
```

```
            throw new BankingException("Transfer amount must be positive");
        }
        if (amount > accountBalance) {
            throw new BankingException("Insufficient funds for transfer");
        }
        this.accountBalance -= amount;
        recipient.accountBalance += amount;
    }

    @Override
    public String toString() {
        return String.format(
            "Customer[ID: %s, Name: %s, Address: %s, Phone: %s, " +
            "SSN: %s, Credit Card: %s, Balance: $%.2f, Created: %s]",
            customerId, fullName, address, phoneNumber,
            getMaskedSSN(), getMaskedCreditCard(), accountBalance, accountCreationDate
        );
    }

    // Detailed string without sensitive data for display
    public String toSecureString() {
        return String.format(
            "Customer Details:\n" +
            "  ID: %s\n" +
            "  Name: %s\n" +
            "  Address: %s\n" +
            "  Phone: %s\n" +
            "  SSN: %s\n" +
            "  Credit Card: %s\n" +
            "  Balance: $%.2f\n" +
            "  Account Created: %s",
            customerId, fullName, address, phoneNumber,
            getMaskedSSN(), getMaskedCreditCard(), accountBalance, accountCreationDate
        );
    }
}

// Encryption utility class
class EncryptionUtil {
    private static final String ALGORITHM = "AES";
    private static final String TRANSFORMATION = "AES/ECB/PKCS5Padding";
    private static final String SECRET_KEY = "ThisIsASecretKey"; // 16 characters for AES-128

    // In production, use proper key management system
    private static SecretKeySpec getSecretKey() {
        return new SecretKeySpec(SECRET_KEY.getBytes(), ALGORITHM);
    }

    public static String encrypt(String data) throws Exception {
        if (data == null) return null;

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.ENCRYPT_MODE, getSecretKey());
        byte[] encryptedBytes = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static String decrypt(String encryptedData) throws Exception {
        if (encryptedData == null) return null;

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.DECRYPT_MODE, getSecretKey());
```

```
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedData));
        return new String(decryptedBytes);
    }

    // Method to encrypt entire serialized object
    public static void encryptAndSerialize(Object obj, String filename) throws Exception {
        // First serialize to byte array
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(obj);
        oos.close();

        byte[] serializedData = baos.toByteArray();

        // Encrypt the serialized data
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.ENCRYPT_MODE, getSecretKey());
        byte[] encryptedData = cipher.doFinal(serializedData);

        // Write encrypted data to file
        FileOutputStream fos = new FileOutputStream(filename);
        fos.write(encryptedData);
        fos.close();

        System.out.println("Object successfully encrypted and serialized to: " + filename);
    }

    // Method to decrypt and deserialize object
    public static Object decryptAndDeserialize(String filename) throws Exception {
        // Read encrypted data from file
        FileInputStream fis = new FileInputStream(filename);
        byte[] encryptedData = fis.readAllBytes();
        fis.close();

        // Decrypt the data
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.DECRYPT_MODE, getSecretKey());
        byte[] decryptedData = cipher.doFinal(encryptedData);

        // Deserialize from decrypted bytes
        ByteArrayInputStream bais = new ByteArrayInputStream(decryptedData);
        ObjectInputStream ois = new ObjectInputStream(bais);
        Object obj = ois.readObject();
        ois.close();

        System.out.println("Object successfully decrypted and deserialized from: " + filename);
        return obj;
    }
}

// Banking service class
class BankingService {
    private static final String DATA_FILE = "customers_encrypted.dat";

    public void saveCustomers(Customer[] customers) {
        try {
            EncryptionUtil.encryptAndSerialize(customers, DATA_FILE);
            System.out.println("âœ“ Successfully saved " + customers.length + " customers with
encryption");
        } catch (Exception e) {
            System.err.println("âœ— Error saving customers: " + e.getMessage());
        }
```

```
    }

    public Customer[] loadCustomers() {
        try {
            File file = new File(DATA_FILE);
            if (!file.exists()) {
                System.out.println("No existing customer data found.");
                return new Customer[0];
            }

            Customer[] customers = (Customer[]) EncryptionUtil.decryptAndDeserialize(DATA_FILE);
            System.out.println("âœ“ Successfully loaded " + customers.length + " customers");
            return customers;
        } catch (Exception e) {
            System.err.println("âœ— Error loading customers: " + e.getMessage());
            return new Customer[0];
        }
    }

    public Customer findCustomerById(Customer[] customers, String customerId) {
        for (Customer customer : customers) {
            if (customer.getCustomerId().equals(customerId)) {
                return customer;
            }
        }
        return null;
    }
}

// Demo class with main method
public class SecureBankingApplication {
    private BankingService bankingService;
    private Customer[] customers;
    private Scanner scanner;

    public SecureBankingApplication() {
        this.bankingService = new BankingService();
        this.scanner = new Scanner(System.in);
        loadOrInitializeData();
    }

    private void loadOrInitializeData() {
        customers = bankingService.loadCustomers();
        if (customers.length == 0) {
            System.out.println("Initializing sample customer data...");
            initializeSampleData();
        }
    }

    private void initializeSampleData() {
        customers = new Customer[]{
            new Customer("CUST001", "John Smith", "123 Main St, New York",
                         "555-0101", "123-45-6789", "password123",
                         "4111111111111111", 5000.00),

            new Customer("CUST002", "Jane Doe", "456 Oak Ave, Los Angeles",
                         "555-0102", "987-65-4321", "securepass456",
                         "5500000000000004", 7500.50),

            new Customer("CUST003", "Bob Johnson", "789 Pine Rd, Chicago",
                         "555-0103", "456-78-9123", "mypassword789",
                         "340000000000009", 3200.75),
```

```
        new Customer("CUST004", "Alice Brown", "321 Elm St, Houston",
                     "555-0104", "789-12-3456", "alicepass321",
                     "6011000000000004", 8900.25),

        new Customer("CUST005", "Charlie Wilson", "654 Maple Dr, Phoenix",
                     "555-0105", "321-54-9876", "charliepass654",
                     "30000000000004", 4500.80)
    };

    bankingService.saveCustomers(customers);
}

public void displayMenu() {
    System.out.println("\n=== SECURE BANKING APPLICATION ===");
    System.out.println("1. View All Customers (Secure Display)");
    System.out.println("2. Find Customer by ID");
    System.out.println("3. Perform Banking Operations");
    System.out.println("4. Save Customer Data");
    System.out.println("5. Reload Customer Data");
    System.out.println("6. Exit");
    System.out.print("Choose an option: ");
}

public void viewAllCustomers() {
    System.out.println("\n=== ALL CUSTOMERS (SECURE DISPLAY) ===");
    for (Customer customer : customers) {
        System.out.println(customer.toSecureString());
        System.out.println("â”€".repeat(50));
    }
}

public void findCustomerById() {
    System.out.print("\nEnter Customer ID: ");
    String customerId = scanner.nextLine();

    Customer customer = bankingService.findCustomerById(customers, customerId);
    if (customer != null) {
        System.out.println("\nâœ“ Customer Found:");
        System.out.println(customer.toSecureString());
    } else {
        System.out.println("âœ— Customer not found with ID: " + customerId);
    }
}

public void performBankingOperations() {
    System.out.print("\nEnter Customer ID: ");
    String customerId = scanner.nextLine();

    Customer customer = bankingService.findCustomerById(customers, customerId);
    if (customer == null) {
        System.out.println("âœ— Customer not found!");
        return;
    }

    System.out.println("\nCustomer: " + customer.getFullName());
    System.out.println("Current Balance: $" + customer.getAccountBalance());

    System.out.println("\nBanking Operations:");
    System.out.println("1. Deposit");
    System.out.println("2. Withdraw");
    System.out.println("3. View Details");
```

```
        System.out.print("Choose operation: ");

        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        try {
            switch (choice) {
                case 1:
                    System.out.print("Enter deposit amount: $");
                    double depositAmount = scanner.nextDouble();
                    scanner.nextLine();
                    customer.deposit(depositAmount);
                    System.out.println("âœ“ Deposit successful! New balance: $" +
customer.getAccountBalance());
                    break;

                case 2:
                    System.out.print("Enter withdrawal amount: $");
                    double withdrawAmount = scanner.nextDouble();
                    scanner.nextLine();
                    customer.withdraw(withdrawAmount);
                    System.out.println("âœ“ Withdrawal successful! New balance: $" +
customer.getAccountBalance());
                    break;

                case 3:
                    System.out.println(customer.toSecureString());
                    break;

                default:
                    System.out.println("âœ— Invalid operation!");
            }
        } catch (BankingException e) {
            System.out.println("âœ— Banking operation failed: " + e.getMessage());
        }
    }

    public void run() {
        boolean running = true;

        while (running) {
            displayMenu();
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    viewAllCustomers();
                    break;
                case 2:
                    findCustomerById();
                    break;
                case 3:
                    performBankingOperations();
                    break;
                case 4:
                    bankingService.saveCustomers(customers);
                    break;
                case 5:
                    loadOrInitializeData();
                    break;
                case 6:
```

```
                    running = false;
                    System.out.println("Thank you for using Secure Banking Application!");
                    break;
                default:
                    System.out.println("âœ— Invalid option! Please try again.");
            }
        }

        scanner.close();
    }

    // Demonstration of the security features
    public static void demonstrateSecurityFeatures() {
        System.out.println("=== SECURITY FEATURES DEMONSTRATION ===");

        try {
            // Test encryption/decryption
            String testData = "Sensitive Customer Data: 123-45-6789";
            System.out.println("Original Data: " + testData);

            String encrypted = EncryptionUtil.encrypt(testData);
            System.out.println("Encrypted Data: " + encrypted);

            String decrypted = EncryptionUtil.decrypt(encrypted);
            System.out.println("Decrypted Data: " + decrypted);

            // Test file encryption
            Customer testCustomer = new Customer("TEST001", "Test User", "Test Address",
                                                 "555-0000", "999-88-7777", "testpass",
                                                 "4111111111111111", 1000.00);

            String testFile = "test_customer.dat";
            EncryptionUtil.encryptAndSerialize(testCustomer, testFile);

            Customer loadedCustomer = (Customer) EncryptionUtil.decryptAndDeserialize(testFile);
            System.out.println("\nLoaded Customer SSN (masked): " +
loadedCustomer.getMaskedSSN());

            // Clean up test file
            new File(testFile).delete();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        System.out.println("ðŸš€ Secure Banking Application Starting...");

        // Demonstrate security features
        demonstrateSecurityFeatures();

        // Start the application
        SecureBankingApplication app = new SecureBankingApplication();
        app.run();
    }
}

// Additional utility for enhanced security (optional)
class SecurityAudit {
    private static final String AUDIT_LOG = "security_audit.log";
```

```
    public static void logAccess(String customerId, String operation) {
        try (FileWriter fw = new FileWriter(AUDIT_LOG, true);
             PrintWriter pw = new PrintWriter(fw)) {
            String timestamp = new Date().toString();
            pw.printf("[%s] Customer: %s, Operation: %s%n", timestamp, customerId, operation);
        } catch (IOException e) {
            System.err.println("Failed to write to audit log: " + e.getMessage());
        }
    }

    public static void logSerialization(String filename, String operation) {
        try (FileWriter fw = new FileWriter(AUDIT_LOG, true);
             PrintWriter pw = new PrintWriter(fw)) {
            String timestamp = new Date().toString();
            pw.printf("[%s] File: %s, Operation: %s%n", timestamp, filename, operation);
        } catch (IOException e) {
            System.err.println("Failed to write to audit log: " + e.getMessage());
        }
    }
}
```