Topic: Object Oriented Thinking

Review

Problem 1. **10 point**

Write a method *removeDuplicates* that takes a *String* and returns a new *String* with duplicate characters removed, while preserving the original order of the characters. For example, given the input *"aabbcc"*, the output should be *"abc"*.

- Use methods such as *indexOf()*, *substring()*.

```
String unique = removeDuplicates("aabbcc");
System.out.println(unique); // Output: "abc"
```

Problem 2. 10 point

Create an **enum** UserRole that represents different roles in a system (ADMIN, MODERATOR, USER). Each role should have an associated permission level. Write a method **getPermissionLevel()** that returns the permission level for each role (e.g., ADMIN has the highest permission, USER has the lowest).

- Define an enum UserRole with the values: ADMIN, MODERATOR, USER.
- Assign each role a permission level (e.g., ADMIN = 3, MODERATOR = 2, USER = 1).
- Add a method getPermissionLevel() to return the permission level.

```
UserRole role = UserRole.MODERATOR;
System.out.println(role.getPermissionLevel()); // Output: 2
```

Problem 3. 10 point

Design and implement a simple ATM system in Java

Class Definitions

- **BankAccount Class:**
  - Properties:
    - accountNumber (String)
    - balance (double)
  - Methods:
    - deposit(double amount): Deposits the specified amount into the account.
    - withdraw(double amount): Withdraws the specified amount from the account.
    - getBalance(): Returns the current balance.
    - getAccountNumber(): Returns the account number.
- **Customer Class:**
  - Properties:
    - name (String)
    - bankAccount (BankAccount)
  - Methods:
    - getName(): Returns the customer's name.

    - * getBankAccount(): Returns the customer's BankAccount.

- **ATM Class:**
  - Properties:
    - * currentCustomer (Customer)
  - Methods:
    - * login(String accountNumber): Logs in the customer by matching the account number.
    - * logout(): Logs out the current customer.
    - * deposit(double amount): Deposits the specified amount into the current customer's account.
    - * withdraw(double amount): Withdraws the specified amount from the current customer's account.
    - * checkBalance(): Returns the current balance of the customer's account.
- **Encapsulation:** Ensure that properties are private and provide public getter and setter methods where appropriate.

Example Usage

Create a main method in a separate class to demonstrate the usage of the ATM system. Follow these steps:

1. **Create Bank Account:**
   - Create a BankAccount instance with the account number set to your student ID and an initial balance of 100,000. (For example, if your student ID is 123456, the account number should be 123456.)

2. **Create Customer:**
   - Create a Customer instance with your first name as the customer's name and associate the previously created BankAccount with this customer.
3. **Instantiate ATM:**
   - Create an ATM instance.
4. **Perform ATM Operations:**
   - Log in using the customer's account number.
   - Perform a deposit of 100,000 into the customer's account.
   - Check the balance of the customer's account to ensure the deposit was successful.
   - Log out the customer.
5. **Code Example:**

```
public class Main {
    public static void main(String[] args) {
        // Create a BankAccount with your student ID and an
          initial balance of 100,000
        BankAccount myAccount = new BankAccount("123456",
          100000);

        // Create a Customer with your first name and associate
          the BankAccount
        Customer customer = new Customer("John", myAccount);
```

```
        // Instantiate the ATM
        ATM atm = new ATM();

        // Log in the customer using their account number
        atm.login(customer.getBankAccount().getAccountNumber());

        // Set the current customer to the ATM (for
          demonstration)
        atm.currentCustomer = customer;

        // Perform deposit of 100,000
        atm.deposit(100000);

        // Check the balance
        System.out.println("Current balance: " +
          atm.checkBalance());

        // Log out the customer
        atm.logout();
    }
}
```

Note: Replace "John" with your actual first name and "123456" with your actual student ID.

Explanation

- **BankAccount Class:** Manages account details and basic transactions (deposit and withdrawal).
- **Customer Class:** Associates a name with a BankAccount.
- **ATM Class:** Handles login, logout, deposits, withdrawals, and balance checks.
- **Main Class:** Demonstrates creating instances and performing operations with the ATM.

Problem 4. 10 point

- Create a TicketMachine object and its methods. You should create the following: getBalance, getPrice, insertMoney, and printTicket.
- Try out the getPrice method. You should see a return value containing the price of the tickets that was set when this object was created.
- Use the insertMoney method to simulate inserting an amount of money into the machine and then use getBalance to check that the machine has a record of the amount inserted. You can insert several separate amounts of money into the machine, just like you might insert multiple coins or notes into a real machine. Try inserting the exact amount required for a ticket. As this is a simple machine, a ticket will not be issued

automatically, so once you have inserted enough money, call the printTicket method.

- A facsimile ticket should be printed in the terminal window.
- What value is returned if you check the machine's balance after it has printed a ticket? Experiment with inserting different amounts of money before printing tickets.
- Do you notice anything strange about the machine's behavior?
- What happens if you insert too much money into the machine – do you receive any refund?
- What happens if you do not insert enough and then try to print a ticket?