**Problem Statement:**

You are tasked with creating a simple banking system in Java that supports account management, transaction operations (deposit, withdrawal, and transfer), and allows querying of account balances. The system will have different types of accounts such as Savings Account, Checking Account, and Business Account.

In this assignment, you will need to incorporate Java 8, Java 11, and Java 17 features for efficient coding and best practices.

**Functional Requirements:**

1. **Account Types**:
   * **Savings Account**: Interest-bearing account with a minimum balance requirement.
   * **Checking Account**: Non-interest-bearing account that allows unlimited withdrawals.
   * **Business Account**: Similar to Checking Account but includes additional features like overdraft protection.
2. **Operations**:
   * **Deposit**: A user can deposit an amount to any account.
   * **Withdrawal**: A user can withdraw an amount from their account (ensuring sufficient balance).
   * **Transfer**: A user can transfer funds between two accounts.
   * **Balance Query**: A user can view the balance of any account.
3. **Transaction History**:
   * Every deposit, withdrawal, or transfer should be recorded in the transaction history of the account.
4. **Interest Calculation**:
   * Implement a method to calculate and apply interest to savings accounts every month.
5. **Account Statement**:
   * A user can get a statement showing their recent transactions for any account.

**Java 8 Features to Use:**

* **Streams**: Use streams to process lists of transactions, accounts, and other objects.
* **Lambda Expressions**: For simplifying the implementation of functional interfaces (e.g., for filtering or sorting).
* **Optional**: Use Optional for handling potentially null values in account-related operations.

**Java 11 Features to Use:**

* **Local-Variable Type Inference (var)**: For defining local variables without explicit type declaration where the type is obvious.
* **String Methods**: Use the new string methods introduced in Java 11 (like isBlank(), lines(), etc.) for more efficient string manipulation.
* **Files API**: Use the updated Files API to read and write transaction history to a file.

**Java 17 Features to Use:**

* **Sealed Classes**: Use sealed classes to define the different types of accounts and restrict subclassing to specific types.
* **Pattern Matching for instanceof**: Use pattern matching with instanceof to simplify type checking.
* **Enhanced Switch Expressions**: Use switch expressions for more concise and expressive handling of account types.

**Suggested Class Structure:**

1. **Account** (abstract class)
   * Fields: accountNumber, balance, transactions
   * Methods: deposit(), withdraw(), getBalance(), addTransaction(), getStatement()
2. **SavingsAccount** (extends Account)
   * Fields: interestRate, minimumBalance
   * Methods: applyInterest()
3. **CheckingAccount** (extends Account)
   * Methods: withdraw() (no minimum balance check)
4. **BusinessAccount** (extends Account)
   * Methods: checkOverdraft() (for handling overdraft protection)
5. **Transaction**
   * Fields: transactionId, date, amount, transactionType
   * Methods: toString()
6. **Bank**
   * Fields: List<Account>
   * Methods: addAccount(), findAccountByNumber(), transfer(), generateAccountStatement()

**Example Use Case:**

1. A user creates a SavingsAccount with an initial deposit.
2. The user can then deposit more funds or withdraw money from the account.
3. Each time a transaction occurs, it's recorded in the transaction history.
4. A monthly interest is applied to the savings account balance.
5. The user can generate an account statement showing all recent transactions.
6. The user can also transfer funds between accounts.