Java Coding Question: Banking Application

Objective:

Create a Java program that simulates a banking application. The application should allow users to create accounts, deposit and withdraw money, view account details, and save all account data to a file. This program should be menu-driven and demonstrate the use of Object-Oriented Programming (OOP) concepts, including inheritance, encapsulation, and polymorphism, as well as file operations and custom exception handling.

Requirements:

1. **Account Class**

   Create an `Account` class that includes the following properties:

   - `accountNumber` (String)
   - `accountHolderName` (String)
   - `balance` (double)

   The class should have:

   - A constructor to initialize the account details.
   - Getter and setter methods for all properties.
   - A method `deposit(double amount)` to deposit money into the account.
   - A method `withdraw(double amount)` to withdraw money from the account. This method should throw a custom exception (`InsufficientBalanceException`) if the balance is insufficient.
   - A method `toString()` to return a string representation of the account details.

2. **SavingsAccount Class**

   Create a subclass `SavingsAccount` that extends `Account`. This class should have:

   - An additional property `interestRate` (double).
   - A constructor to initialize all properties including those from the `Account` class.
   - Override the `withdraw(double amount)` method to allow withdrawals only if the balance after the withdrawal remains above a minimum balance (e.g., $100).

3. **BankingApplication Class**
   The `BankingApplication` class should provide a menu-driven interface that allows users to perform the following operations:
   - **Create a new account:** Prompt the user to enter account details and save them in a list.
   - **Deposit money:** Allow the user to deposit money into an existing account.
   - **Withdraw money:** Allow the user to withdraw money from an existing account. Handle `InsufficientBalanceException` if the withdrawal amount is more than the current balance.
   - **View account details:** Display details of an account based on the account number.
   - **Delete an account:** Remove an account from the list based on the account number.
   - **Save accounts to file:** Save all account details to a file (`accounts.txt`).
   - **Load accounts from file:** Load account details from a file (`accounts.txt`) and display them.
4. **Custom Exceptions**
   - Create a custom exception class `InsufficientBalanceException` that should be thrown when a withdrawal amount exceeds the available balance.
5. **File Operations**
   - Implement methods to save account details to a file and load account details from a file. Use appropriate exception handling for file operations.

Method Descriptions:

1. `void createAccount()`
   Prompts the user to enter account details (account number, account holder name, initial balance, and interest rate for savings account) and creates either a `SavingsAccount` or `Account` object based on user input. Adds the account to a list of accounts.
2. `void depositMoney(String accountNumber, double amount)`
   Finds the account by `accountNumber` from the list, deposits the specified `amount` using the `deposit(double amount)` method of the `Account` class.
3. `void withdrawMoney(String accountNumber, double amount)`
   Finds the account by `accountNumber` from the list, withdraws the specified `amount` using the `withdraw(double amount)` method of the `Account` class. Handles `InsufficientBalanceException` to provide a user-friendly error message.
4. `void viewAccountDetails(String accountNumber)`
   Finds the account by `accountNumber` and displays its details using the `toString()` method.
5. `void deleteAccount(String accountNumber)`
   Removes the account from the list by `accountNumber`.
6. `void saveAccountsToFile()`
   Writes all account details to a file (`accounts.txt`). Uses file I/O operations and handles exceptions related to file writing.
7. `void loadAccountsFromFile()`
   Reads account details from a file (`accounts.txt`) and displays them. Uses file I/O operations and handles exceptions related to file reading.
8. `void displayMenu()`
   Displays the menu options to the user and takes input to perform the selected operation.

Sample Menu:

markdown Kodu kopyala

```
Welcome to the Banking Application
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice:
```

Sample Output:

markdown Kodu kopyala

```
Welcome to the Banking Application
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice: 1

Enter Account Number: 12345
Enter Account Holder Name: John Doe
Enter Initial Balance: 5000
Is this a savings account? (yes/no): yes
Enter Interest Rate: 2.5

Account created successfully!

1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice: 2

Enter Account Number: 12345
Enter Deposit Amount: 1000

Deposit successful!

1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice: 3

Enter Account Number: 12345
Enter Withdraw Amount: 6000

Insufficient Balance! Cannot withdraw the specified amount.
```

```
1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice: 4

Enter Account Number: 12345

Account Details:
Account Number: 12345
Account Holder Name: John Doe
Account Balance: 5000
Interest Rate: 2.5

1. Create a new account
2. Deposit money
3. Withdraw money
4. View account details
5. Delete an account
6. Save accounts to file
7. Load accounts from file
8. Exit
Enter your choice: 8

Thank you for using the Banking Application!
```

Notes:

- Ensure that all classes, methods, and custom exceptions are properly documented with comments.
- Use appropriate OOP principles to ensure code modularity and reusability.
- Handle all possible exceptions, including input mismatches and file operation errors, to make the application robust.