Java Bank Accounts Simulator using Object Oriented Programming

The Bank Account Simulation example covers most Object Oriented Programming features i.e. Class, Object, Inheritance, Polymorphism, Encapsulation, etc.

BankAccount Blueprint and Template

1. **State / Attributes**
2. accountName
3. accountNumber
4. balance
5. **Behaviors / Methods**
6. BankAccount(String accNumber, String accName)
7. getAccountName()
8. getAccountNumber()
9. getBalance()
10. deposit(double amount)
11. withdraw(double amount)

BankAccount Demo

Following BankAccountDemo.java demonstrates the use of BankAccount.java

```
public class BankAccountDemo {

   public static void main(String[] args) {
        BankAccount absherzad = new BankAccount("20120", "Abdul Rahman Sherzad");

        absherzad.deposit(500);
        absherzad.deposit(1500);

        System.out.println("Balance is: " + absherzad.getBalance()); // 2000
        absherzad.withdraw(400);

        System.out.println("Balance is: " + absherzad.getBalance()); // 1600
    }
}
```

Answer

```
public class BankAccount {
	// BankAccount attributes
	private String accountNumber;
	private String accountName;
	private double balance;

	// BankAccount methods

	/**
	 * This is the constructor responsible for account creation with
initial
	 * balance 0.0
	 * 
	 * @param accNumber
	 *            Bank Account Number as String
	 * @param accName
	 *            Bank Account Name as String
	 */
	public BankAccount(String accNumber, String accName) {
		accountNumber = accNumber;
		accountName = accName;
		balance = 0;
	}

	// methods to read the attributes

	/**
	 * Returns the Account Name of the bank account object.
	 * 
	 * @return accountName
	 */
	public String getAccountName() {
		return accountName;
	}

	/**
	 * Returns the Account Number of the bank account object.
	 *
	 * @return accountNumber
	 */
	public String getAccountNumber() {
		return accountNumber;
	}

	/**
	 * Returns the Balance value of the bank account object.
	 *
	 * @return balance
	 */
	public double getBalance() {
		return balance;
	}

	/**
	 * This method take care of the deposit transaction Return true
on success
	 * and false on failure
	 *
	 * @param amount
	 *            the amount to be deposited
	 * @return boolean
	 */
	public boolean deposit(double amount) {
		if (amount > 0) {
			balance = balance + amount;
			return true;
		} else {
			return false;
		}
	}

	/**
	 * This method take care of the withdraw transaction Return true
on success
	 * and false on failure
	 *
	 * @param amount
	 *            the amount to be withdrawn
	 * @return boolean
	 */
	public boolean withdraw(double amount) {
		if (amount > balance) {
			return false;
```

```
            } else {
                balance = balance - amount;
                return true;
            }
        }
}
```

SavingsAccount Blueprint and Template

1. **State / Attributes**
2. interestRate
3. accountName *// inherited from BankAccount*
4. accountNumber *// inherited from BankAccount*
5. balance *// inherited from BankAccount*
6. **Behaviors / Methods**
7. SavingsAccount(String accNumber, String accName, double rate)
8. addInterest()
9. BankAccount(String accNumber, String accName) *// inherited from BankAccount*
10. getAccountName() *// inherited from BankAccount*
11. getAccountNumber() *// inherited from BankAccount*
12. getBalance() *// inherited from BankAccount*
13. deposit(double amount) *// inherited from BankAccount*
14. withdraw(double amount) *// inherited from BankAccount*

SavingsAccount Demo

Following SavingsAccountDemo.java demonstrates the use of SavingsAccount.java

```
public class SavingsAccountDemo {

   public static void main(String[] args) {
        SavingsAccount saving = new SavingsAccount("20120",
                        "Abdul Rahman Sherzad", 10);
        saving.deposit(500);
        System.out.println("Balance Before Interest: " + saving.getBalance());

        saving.addInterest();
        System.out.println("Balance After Interest: " + saving.getBalance());
    }
}
```

Answer

```
ublic class SavingsAccount extends BankAccount {
	private double interestRate;

	/**
	 * The SavingsAccount constructor is responsible creating
SavingsAccount and
	 * in the meanwhile calling the BankAccount constructor to create
the Bank
	 * Account with given Account Number and Account Name
	 *
	 * @param accNumber
	 *            Bank Account# as String
	 * @param accName
	 *            Bank Account Name as String
	 * @param rate
	 *            Interest Rate as double
	 */
	public SavingsAccount(String accNumber, String accName, double
rate) {
		super(accNumber, accName);
		interestRate = rate;
	}

	/**
	 * interest is calculated and added to the balance by calling the
deposit()
	 * method of parent class periodically
	 */
	public void addInterest() {
		double interest = getBalance() * interestRate / 100;
		deposit(interest);
	}

}
```

CheckingAccount Blueprint and Template

1. **State / Attributes**
2. transactionCount
3. NUM_FREE
4. TRANS_FEE
5. accountName // *inherited from BankAccount*
6. accountNumber // *inherited from BankAccount*
7. balance // *inherited from BankAccount*
8. **Behaviors / Methods**
9. CheckingAccount(String accNumber, String accName)
10. BankAccount(String accNumber, String accName) // *inherited from BankAccount*
11. getAccountName() // *inherited from BankAccount*
12. getAccountNumber() // *inherited from BankAccount*
13. getBalance() // *inherited from BankAccount*
14. deductFees()
15. deposit(double amount) // *Overridden*
16. withdraw(double amount) // *Overridden*
17. deposit(double amount) // *inherited from BankAccount*
18. withdraw(double amount) // *inherited from BankAccount*

CheckingAccount Demo

Following CheckingAccountDemo.java demonstrates the use of CheckingAccount.java

```
public class CheckingAccountDemo {
   public static void main(String[] args) {
        CheckingAccount checking = new CheckingAccount("20120",
                        "Abdul Rahman Sherzad");

          checking.deposit(500);
          checking.withdraw(200);
          checking.deposit(700);
          // No deduction fee because we had only 3 transactions
          checking.deductFees();
          System.out.println("transactions <= 3: " + checking.getBalance());

          // One more transaction
          checking.deposit(200);
          // Deduction fee occurs because we have had 4 transactions
          checking.deductFees();
          System.out.println("transactions > 3: " + checking.getBalance());
     }
```

Answer

```
public class CheckingAccount extends BankAccount {
	private int transactionCount;
	private static final int NUM_FREE = 3;
	private static final double TRANS_FEE = 2.0;

	/**
	 * The CheckingAccount constructor is responsible creating
CheckingAccount
	 * by calling the BankAccount constructor to create the Bank
Account with
	 * given Account Number and Account Name
	 * 
	 * @param accNumber
	 *            Account Number as String
	 * @param accName
	 *            Account Name as String
	 */
	public CheckingAccount(String accNumber, String accName) {
		super(accNumber, accName);
		transactionCount = 0;
	}

	/**
	 * Overridden deposit() method tracking the number of
transactions
	 */
	public boolean deposit(double amount) {
		if (super.deposit(amount)) {
```

```
                transactionCount++;
                return true;
            }
            return false;
        }

        /**
         * Overridden withdraw() method tracking the number of
transactions
         */
        public boolean withdraw(double amount) {
            if (super.withdraw(amount)) {
                transactionCount++;
                return true;
            }
            return false;
        }

        /**
         * calculates the deduction fee with a fee charged for subsequent
         * transactions and deduct it from the balance by calling its
Parent Class
         * withdraw() method as well as reset the transactionCount
         */
        public void deductFees() {
            if (transactionCount > NUM_FREE) {
                double fees = TRANS_FEE * (transactionCount -
NUM_FREE);
                if (super.withdraw(fees)) {
                    transactionCount = 0;
                }
            }
        }
}
```