**Java Banking Assignment: Bank Account Management System**

**Problem Statement:**

You are tasked with developing a **Bank Account Management System** that allows multiple users to perform various banking operations like deposit, withdraw, transfer funds, and check balance. The system must use **Object-Oriented Programming (OOP)** principles, **Collections**, **Exceptions**, and **Multithreading** to ensure smooth operation and efficiency in handling multiple users simultaneously.

**Functional Requirements:**

1. **Bank Account Class:**
   * Represent a bank account with attributes such as account number, account holder’s name, account balance, and account type.
   * Provide methods to deposit money, withdraw money, transfer money, and check balance.
2. **Bank Class:**
   * Store and manage a collection of bank accounts.
   * Allow searching for a bank account by account number.
   * Ensure that multiple users can interact with their accounts concurrently.
3. **Transaction Class:**
   * Represent a transaction (deposit, withdraw, or transfer).
   * Handle the details of a transaction, including time, type, amount, and the associated account.
4. **Exception Handling:**
   * Handle errors such as insufficient funds, invalid account details, or invalid transaction type through custom exceptions.
5. **Multithreading:**
   * Use multithreading to simulate concurrent banking operations, such as multiple customers trying to perform transactions on their accounts simultaneously.
6. **Collections:**
   * Use a collection such as **HashMap** or **ArrayList** to manage multiple bank accounts.
   * Use a **Queue** to simulate the transaction processing system where transactions are handled one at a time.

**Detailed Breakdown:**

1. **BankAccount Class:**
   * Attributes:
     + accountNumber (String)
     + accountHolderName (String)
     + balance (double)
     + accountType (String - "Saving" or "Current")
   * Methods:
     + deposit(double amount)
     + withdraw(double amount)
     + transfer(BankAccount targetAccount, double amount)
     + checkBalance()
   * Include a method to check the account balance.
   * Ensure that withdrawals do not exceed the available balance.
2. **Bank Class:**
   * Attributes:
     + accountsMap (HashMap<String, BankAccount>) - to hold all accounts indexed by account number.
   * Methods:
     + addAccount(BankAccount account)
     + getAccount(String accountNumber) - retrieves an account by its number.
     + processTransaction(Transaction transaction) - to simulate the transaction process.
3. **Transaction Class:**
   * Attributes:
     + transactionID (String)
     + transactionType (String - "Deposit", "Withdraw", "Transfer")
     + amount (double)
     + accountNumber (String)
     + timestamp (LocalDateTime)
   * Methods:
     + process() - processes the transaction.
4. **Custom Exceptions:**
   * InsufficientFundsException: Raised when a withdrawal or transfer exceeds the available balance.
   * InvalidAccountException: Raised when an invalid account number is provided.
   * InvalidTransactionException: Raised when an invalid transaction type is provided (other than "Deposit", "Withdraw", "Transfer").
5. **Multithreading:**
   * Use a thread pool to handle multiple transactions concurrently.
   * Use synchronized blocks or other thread-safety mechanisms to ensure that account balance updates are atomic.
   * Use ExecutorService to manage threads in processing transactions.
   * Ensure that transactions are processed in sequence (first-come, first-served).