Banking System

- The following **Directory structure** is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface/abstract class to showcase functionalities.
    - Create the implementation class for the above interface/abstract class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a **DBPropertyUtil** class with a static function which takes property file name as parameter and returns connection string.
    - Create a **DBConnUtil** class which holds **static method** which takes connection string as parameter file and returns **connection object**.
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

OOPS, Collections and Exception Handling

Task 7: Class & Object

1. Create a `Customer` class with the following confidential attributes:
   - Attributes
     - Customer ID
     - First Name
     - Last Name
     - Email Address
     - Phone Number
     - Address
   - Constructor and Methods
     - Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

2. Create an `Account` class with the following confidential attributes:
   - Attributes
     - Account Number
     - Account Type (e.g., Savings, Current)
     - Account Balance
   - Constructor and Methods
     - Implement default constructors and overload the constructor with Account attributes,
     - Generate getter and setter, (print all information of attribute) methods for the attributes.
     - Add methods to the `Account` class to allow deposits and withdrawals.
       - deposit(amount: float): Deposit the specified amount into the account.

- withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
- calculate_interest(): method for calculating interest amount for the available balance. interest rate is fixed to 4.5%

- Create a Bank class to represent the banking system. Perform the following operation in main method:
  - create object for account class by calling parameter constructor.
  - deposit(amount: float): Deposit the specified amount into the account.
  - withdraw(amount: float): Withdraw the specified amount from the account.
  - calculate_interest(): Calculate and add interest to the account balance for savings accounts.

Create **ICustomerServiceProvider** interface/abstract class with following functions:

- **get_account_balance(account_number: long)**: Retrieve the balance of an account given its account number. should return the current balance of account.
- **deposit(account_number: long, amount: float)**: Deposit the specified amount into the account. Should return the current balance of account.
- **withdraw(account_number: long, amount: float)**: Withdraw the specified amount from the account. Should return the current balance of account. A savings account should maintain a minimum balance and checking if the withdrawal violates the minimum balance rule.
- **transfer(from_account_number: long, to_account_number: int, amount: float)**: Transfer money from one account to another.
- **getAccountDetails(account_number: long):** Should return the account and customer details.

Create **IBankServiceProvider** interface/abstract class with following functions:

- **create_account(Customer customer, long accNo, String accType, float balance)**: Create a new bank account for the given customer with the initial balance.
- **listAccounts()**:Account[] accounts: List all accounts in the bank.

- **calculateInterest():** the calculate_interest() method to calculate interest based on the balance and interest rate.

Create a Bank Class and must have following requirements:

1. Create a Bank class to represent the banking system. It should have the following methods:
    - **create_account(Customer customer, long accNo, String accType, float balance)**: Create a new bank account for the given customer with the initial balance.
    - **get_account_balance(account_number: long)**: Retrieve the balance of an account given its account number. should return the current balance of account.
    - **deposit(account_number: long, amount: float)**: Deposit the specified amount into the account. Should return the current balance of account.
    - **withdraw(account_number: long, amount: float)**: Withdraw the specified amount from the account. Should return the current balance of account.
    - **transfer(from_account_number: long, to_account_number: int, amount: float)**: Transfer money from one account to another.
    - **getAccountDetails(account_number: long):** Should return the account and customer details.

Task 12: Exception Handling

throw the exception whenever needed and Handle in main method,

1. **InsufficientFundException** throw this exception when user try to withdraw amount or transfer amount to another account and the account runs out of money in the account.
2. **InvalidAccountException** throw this exception when user entered the invalid account number when tries to transfer amount, get account details classes.
3. **OverDraftLimitExcededException** thow this exception when current account customer try to with draw amount from the current account.
4. **NullPointerException** handle in main method**.**

Throw these exceptions from the methods in HMBank class. Make necessary changes to accommodate these exception in the source code. Handle all these exceptions from the main program.