Trade Hub Documentation

Overview

Trade Hub is a console-based trading application that simulates the buying of assets such as stocks. Users can create accounts, manage their portfolios, and perform trades using a virtual trading account.

Table of Contents

Features

- **User Account Management**: Users can create an account with their name, email, and phone number.
- **Portfolio Management**: Users can view and manage their collection of purchased assets.
- **Trading Account**: Allows users to add funds and perform trades.
- **Order Placement**: Users can purchase assets by specifying the desired quantity.

How It Works

1. **User Registration**: Users provide their details to create an account.
2. **Add Funds**: Users deposit funds into their trading account.
3. **View Assets**: Users can view a list of available assets for trading.
4. **Place Orders**: Users select an asset, specify a quantity, and place an order.
5. **Manage Portfolio**: Users can view their purchased assets and quantities.

Code Components

- `Asset` **Interface**: Blueprint for asset classes.
- `StockAsset` **Class**: Represents stock-based assets.
- `Order` **Class**: Represents a trade order.
- `Portfolio` **Class**: Manages user's collection of assets.
- `TradingAccount` **Class**: Handles account balance and transactions.
- `User` **Class**: Represents a user.
- `UserService` **Class**: Manages user-related operations.
- `PortfolioService` **Class**: Handles portfolio-related operations.
- `Main` **Class**: Entry point of the application.

Class Descriptions

1. `Asset` Interface

Defines the structure for all asset types.

- **Methods**:
  - `String getId()`: Returns the asset's ID.
  - `String getName()`: Returns the asset's name.
  - `double getPrice()`: Returns the asset's price.

2. `StockAsset` Class

Represents stocks as tradable assets.

- **Attributes**:
  - `String assetId`: Unique identifier.
  - `String name`: Name of the stock.
  - `double price`: Price of the stock.
- **Implements Methods**:
  - `getId()`, `getName()`, `getPrice()`.

3. `Order` Class

Tracks a single trade order.

- **Attributes**:
  - `String orderId`: Unique identifier for the order.
  - `Asset asset`: Asset being traded.
  - `int quantity`: Quantity purchased.
- **Methods**:
  - `Asset getAsset()`: Returns the asset.
  - `int getQuantity()`: Returns the quantity ordered.

4. `Portfolio` Class

Manages a collection of orders.

- **Attributes**:
  - `List<Order> orders`: List of all orders.
- **Methods**:
  - `addOrder(Order order)`: Adds an order to the portfolio.
  - `viewPortfolio()`: Displays all assets and their quantities.

5. `TradingAccount` Class

Handles user's trading funds.

- **Attributes**:
  - `double balance`: Available funds.
- **Methods**:
  - `addFunds(double amount)`: Adds funds to the account.
  - `boolean deductFunds(double amount)`: Deducts funds for a transaction, returns `false` if insufficient.
  - `getBalance()`: Returns the current balance.

6. `User` Class

Stores user information.

- **Attributes**:
  - `String userId`: Unique identifier.
  - `String name`: Name of the user.
  - `String email`: Email address.
  - `String phone`: Phone number.
- **Methods**:
  - `getName()`: Returns the user's name.

7. `UserService` Class

Manages user-related operations.

- **Methods**:
  - `createUser(String userId, String name, String email, String phone)`: Creates and returns a new user.

8. `PortfolioService` Class

Handles portfolio operations.

- **Methods**:
  - `placeOrder(User user, Portfolio portfolio, TradingAccount account, Asset asset, int quantity)`: Places an order if funds are sufficient.

9. `Main` Class

The main execution point of the program.

- **Flow**:
  1. User registration.
  2. Add funds to trading account.
  3. Display available assets.
  4. Place an order.
  5. Display the user's portfolio.

Execution Flow

1. **Initialize Services**: `UserService` and `PortfolioService` are instantiated.
2. **Collect User Input**: Name, email, phone, and initial funds are collected.
3. **Display Assets**: Available assets are displayed with prices.
4. **Asset Selection**: User selects an asset and specifies the quantity.
5. **Place Order**: Funds are deducted, and the order is added to the portfolio.
6. **View Portfolio**: User views their purchased assets.

Future Enhancements

1. **Persistent Storage**:
   - Save user data, portfolio, and transactions in a database.
2. **Asset Management**:
   - Add more asset types (e.g., bonds, ETFs).
3. **Live Data Integration**:
   - Fetch real-time asset prices using APIs.
4. **User Authentication**:
   - Add login and password-based authentication.
5. **Enhanced UI**:
   - Replace the console interface with a GUI or web interface.
6. **Reporting**:
   - Add profit/loss tracking and reporting features.

This documentation serves as a complete guide to understanding and working with the Trade Hub application.