Simple Banking Application

By Ramesh Fadatare (Java Guides)

Banking Application Overview

This Spring Boot banking application serves as a simple yet functional platform for managing bank accounts and performing basic banking operations.

The application supports creating bank accounts, fetching account details, making deposits and withdrawals, deleting accounts, transferring funds between accounts, and viewing transaction histories.

Requirement 1 - Account Management

Account Management feature allows User to create account, get specific account details, get all accounts, withdraw amount, deposit amount, and delete amount.

Build Account REST APIs:

1. **Create Account**: Create a new Account
2. **Get Account**: Retrieve a specific account details
3. **Get All Accounts**: Retrieve all accounts
4. **Withdraw Amount**: Withdraw amount from the account
5. **Deposit Amount**: Deposit amount to the Account
6. **Delete Account**: Delete the existing account

Requirement 2 - Exception Handling

The REST APIs should return the proper error response as shown below:

```
{
    "timestamp": "2024-03-06T20:37:34.448725",
    "message": "Account does not exists",
    "details": "uri=/api/accounts/3/deposit",
    "errorCode": "ACCOUNT_NOT_FOUND"
}
```

Requirement 3 - Transferring Funds Between Accounts

This feature involves debiting an amount from one account and crediting it into another.

Transfer Fund REST API: To transfer amount from one account to another account.

Development Steps

1. Create TransferFundDto record class
2. Define transferFunds method in AccountService interface
3. Implement transferFunds method in AccountServiceImpl class
4. Build /transfer REST API
5. Test /transfer REST API using Postman Client

Requirement 4 - Transaction History Management

This feature maintains the transaction of every deposit, withdrawal, and transfer operations.

Log the Transactions for DEPOSIT, WITHDRAW, and TRANSFER

Fetch Transaction History: Get an account's transaction history

Development Steps

1. Create Transaction Entity
2. Create TransactionRepository and Define query method to retrieve list of transactions by account id order by descending order (newest to oldest)
3. Change AccountServiceImpl.deposit() method to log transaction for DEPOSIT
4. Change AccountServiceImpl.withdraw() method to log transaction for WITHDRAW
5. Change AccountServiceImpl.transferFunds() method to log transaction for TRANSFER
6. Create TransactionDto record
7. Implement getAccountTransactions method in AccountServiceImpl class
8. Create /**{accountId}/transactions** REST API
9. Test /**{accountId}/transactions** REST API using Postman Client

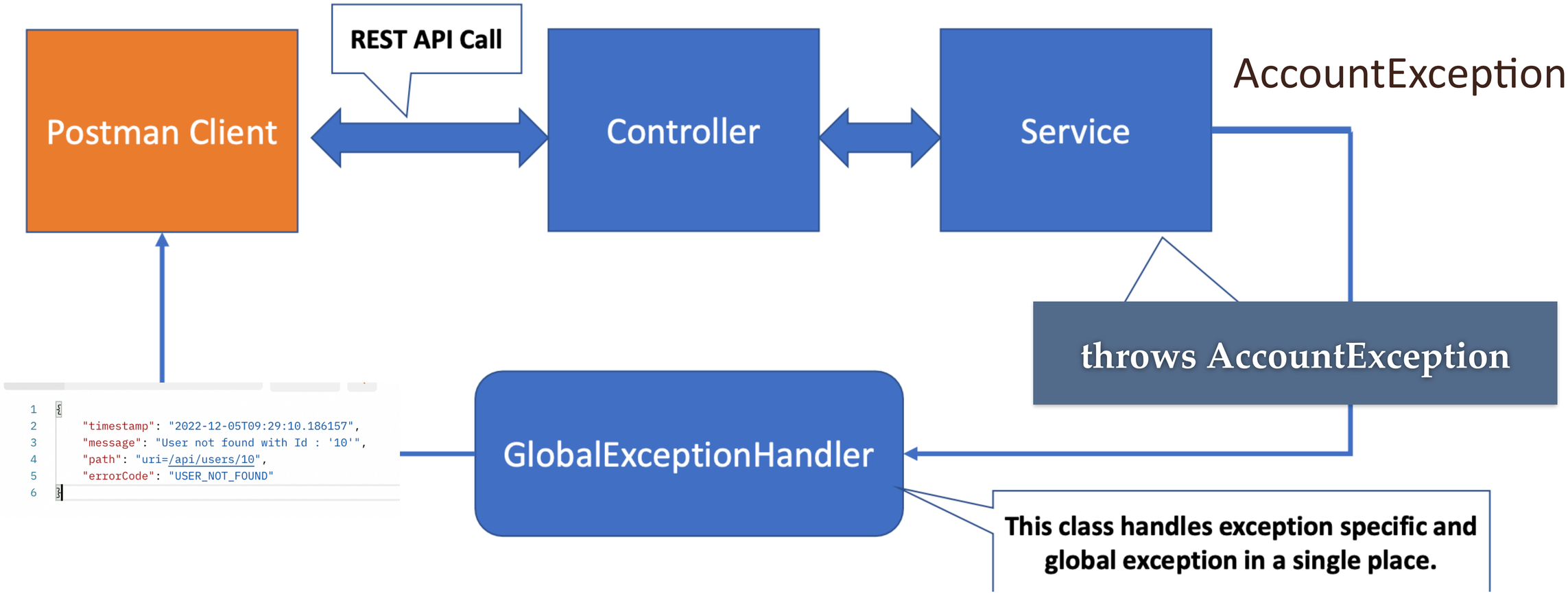
Using Record Class as DTO

1. Java Record is a special kind of class that helps you encapsulate related data without the need for boilerplate code.
2. Using a Record class as a DTO (Data Transfer Object) in a Spring Boot application is a modern and efficient approach to encapsulating data transfer between the application layers.
3. Records are a good fit for DTOs because they are concise, immutable, and automatically provide implementations of getter(), constructors, equals(), hashCode(), and toString() methods, which are essential for DTOs. (reducing boilerplate code)

Exception Handling in Banking App

By Ramesh Fadatare (Java Guides)

Spring Boot REST API Exception Handling



Exception Handling in Spring Boot App

1. Handling exceptions in Spring Boot REST APIs is typically done using the **@ControllerAdvice** and **@ExceptionHandler** annotations.

2. **@ControllerAdvice** enables global exception handling for controllers.

3. **@ExceptionHandler** defines methods to handle specific exceptions within a controller or globally with @ControllerAdvice.

Development Steps

1. Create custom exception named **AccountException**
2. Using **AccountException**
3. Create **ErrorDetails** Class to hold error response
4. Create **GlobalExceptionHandler** to handle specific exceptions as well as generic exceptions