Bank Project Task

Sayali B Kale.

sayalikaleofficial@gmail.com

GitHub UserID : Sayali-Kale-Official

Repository Link: Sayali-Kale-Official/DWS-Challenge

Task Given :

Your task is to add functionality for a transfer of money between accounts.

Transfers should be specified by providing:

1] accountFrom id ; 2] accountTo id ; 3] amount to transfer between accounts

The amount to transfer should always be a positive number. It should not be possible for an account to end up with negative balance (we do not support overdrafts!)

Whenever a transfer is made, a notification should be sent to both account holders, with a message containing id of the other account and amount transferred.

==========

Changes I've Done:

End Points to check:

1] Post: Create Account.

http://localhost:8080/v1/accounts

```
{
  "accountId": "Id-2",
  "balance": 2000
}
{
  "accountId": "Id-1",
  "balance": 1000
}
```

2] Get: Get Account Details.

http://localhost:8080/v1/accounts/Id-1

http://localhost:8080/v1/accounts/Id-2

3] Post: Transfer Money.

http://localhost:8080/v1/accounts/transfer

```
{
  "fromAccountId": "Id-2",
  "toAccountId": "Id-1",
  "amount": 100
}
```

Output Sample :

```
: Completed initialization in 1 ms
: Creating account Account(accountId=Id-1, balance=1000)
: Creating account Account(accountId=Id-2, balance=2000)
: Retrieving account for id Id-2
: Retrieving account for id Id-1
: Sending notification to owner of Id-2: Transferred 100 to account Id-1
: Sending notification to owner of Id-1: Received 100 from account Id-2
: Retrieving account for id Id-1
```

Updated Balance:

1] GET : http://localhost:8080/v1/accounts/Id-1

Postman Response:

```
{
   "accountId": "Id-1",
   "balance": 1100
}
```

2] GET: http://localhost:8080/v1/accounts/Id-2

Postman Response:

```
{
   "accountId": "Id-2",
   "balance": 1900
}
```

1] I’ve Created A Model Class by the name: Transfer Details

Here is the code:

```
@Data
public class TransferDetails {
@NotNull(message = "Source account ID is required")
private String fromAccountId;
@NotNull(message = "Destination account ID is required")
private String toAccountId;
@NotNull(message = "Transfer amount is required")
@Min(value = 1, message = "Transfer amount must be greater than zero")
private BigDecimal amount;
}
```

==========

2] I’ve Created a Method in controller class : transferMoney Method.

```
@PostMapping("/transfer")
public ResponseEntity<String> transferMoney(@RequestBody @Valid TransferDetails transferdetails) {
try {
accountsService.transferMoney(
transferdetails.getFromAccountId(),
transferdetails.getToAccountId(),
transferdetails.getAmount()
);
return ResponseEntity.ok("Transfer successful");
} catch (IllegalArgumentException e) {
return ResponseEntity.badRequest().body(e.getMessage());
}
}
```

==========

3] I’ve Created several methods inside Service Class to implement Transfer Money Operation logic.

Code:

```
public void transferMoney(String fromAccountId, String toAccountId, BigDecimal transferAmount) {
```

```
if (transferAmount.compareTo(BigDecimal.ZERO) <= 0) {
throw new IllegalArgumentException("Enter a Valid Positive Transfer Amount");
}
Account accountFrom = accountsRepository.getAccount(fromAccountId);
Account accountTo = accountsRepository.getAccount(toAccountId);
withdraw(accountFrom,transferAmount);
deposit(accountTo, transferAmount);
// Create the transfer description for the notification
String transferDescriptionFrom = String.format("Transferred %s to account %s", transferAmount, toAccountId);
notificationService.notifyAboutTransfer(accountFrom, transferDescriptionFrom);
String transferDescriptionTo = String.format("Received %s from account %s", transferAmount, fromAccountId);
notificationService.notifyAboutTransfer(accountTo, transferDescriptionTo);
}
// Withdraw method to decrease the balance
public synchronized void withdraw(Account accountFrom, BigDecimal withdrawAmount) {
BigDecimal balance = accountFrom.getBalance();
if (balance.compareTo(withdrawAmount) < 0) {
throw new IllegalArgumentException("Insufficient balance");
}
balance = balance.subtract(withdrawAmount);
accountFrom.setBalance(balance);
}
// Deposit method to increase the balance
public synchronized void deposit(Account accountTo, BigDecimal depositAmount) {
BigDecimal balance = accountTo.getBalance();
balance = balance.add(depositAmount);
accountTo.setBalance(balance);
}
```

==========

4] I’ve Created a Account Service Test Class to write Junit test cases :

```
@SpringBootTest
class AccountsServiceTest {
```

```
@MockBean
private AccountsRepository accountsRepository;
@MockBean
private NotificationService notificationService;
private AccountsService accountsService;
@BeforeEach
void setUp() {
// Initialize mocks for each test
notificationService = mock(NotificationService.class);
accountsRepository = mock(AccountsRepository.class);
// Manually inject mocks into AccountsService
accountsService = new AccountsService(accountsRepository, notificationService);
}
@Test
void createAccount_success() {
Account account = new Account("Id-1", BigDecimal.valueOf(1000));
accountsService.createAccount(account);
verify(accountsRepository, times(1)).createAccount(account);
}
@Test
void getAccount_success() {
Account account = new Account("Id-1", BigDecimal.valueOf(1000));
when(accountsRepository.getAccount("Id-1")).thenReturn(account);
Account retrievedAccount = accountsService.getAccount("Id-1");
assertThat(retrievedAccount).isEqualTo(account);
verify(accountsRepository, times(1)).getAccount("Id-1");
}
@Test
void transferMoney_success() {
Account accountFrom = new Account("Id-1", BigDecimal.valueOf(1000));
Account accountTo = new Account("Id-2", BigDecimal.valueOf(500));
when(accountsRepository.getAccount("Id-1")).thenReturn(accountFrom);
when(accountsRepository.getAccount("Id-2")).thenReturn(accountTo);
accountsService.transferMoney("Id-1", "Id-2", BigDecimal.valueOf(200));
```

```
assertThat(accountFrom.getBalance()).isEqualByComparingTo("800");
assertThat(accountTo.getBalance()).isEqualByComparingTo("700");
verify(notificationService, times(1)).notifyAboutTransfer(accountFrom, "Transferred 200 to account Id-2");
verify(notificationService, times(1)).notifyAboutTransfer(accountTo, "Received 200 from account Id-1");
}
@Test
void transferMoney_insufficientBalance() {
Account accountFrom = new Account("Id-1", BigDecimal.valueOf(100));
Account accountTo = new Account("Id-2", BigDecimal.valueOf(500));
when(accountsRepository.getAccount("Id-1")).thenReturn(accountFrom);
when(accountsRepository.getAccount("Id-2")).thenReturn(accountTo);
assertThrows(IllegalArgumentException.class,
() -> accountsService.transferMoney("Id-1", "Id-2", BigDecimal.valueOf(200)));
assertThat(accountFrom.getBalance()).isEqualByComparingTo("100");
assertThat(accountTo.getBalance()).isEqualByComparingTo("500");
verify(notificationService, never()).notifyAboutTransfer(any(), any());
}
@Test
void withdraw_success() {
Account account = new Account("Id-1", BigDecimal.valueOf(1000));
accountsService.withdraw(account, BigDecimal.valueOf(300));
assertThat(account.getBalance()).isEqualByComparingTo("700");
}
@Test
void withdraw_insufficientBalance() {
Account account = new Account("Id-1", BigDecimal.valueOf(100));
assertThrows(IllegalArgumentException.class, () -> accountsService.withdraw(account, BigDecimal.valueOf(200)));
assertThat(account.getBalance()).isEqualByComparingTo("100");
}
@Test
void deposit_success() {
Account account = new Account("Id-2", BigDecimal.valueOf(500));
```

```
accountsService.deposit(account, BigDecimal.valueOf(200));

assertThat(account.getBalance()).isEqualByComparingTo("700");

}
```

==========

Conclusion:

1] I've created a New Model class by the name Transfer Details with 3 variables as mentioned in the task.

2] Using those I tried creating a new method in controller class to transfer money & mapped it with existing class & service class accordingly.

3] Then I've implemented the Transfer Money Logic in service class using the Method.

4] Then to transfer the money from 1 Account to Another Ive created 2 more methods with names Withdraw & deposit & implemented the logic there itself.

5] Then I mapped the class to AccountRepository to store it in it.

6] Similarly, I've also tried to print the logs as mentioned in the task rules using Email Notification Repository Without changing anything.

7] Finally, I write the Junit test cases in the class named as Account Service test & tried to remove the dependency of that class with other repositories using @Mock & @MockBean Annotations.

Note:

1] In the task it was mentioned to use gradle as a build tool. But I've used Maven because of my existing setup & I'm more used to the maven compared to gradle.

2] Also as an Addon to the existing task, Ive tried to implement the logic to print the updated balance of both the account holders on the same Log Message where money & account details are now printing, But then I realised that I was told not to modify any things into the EmailNotificationService Class, So I left that as it is.

3] Also Ive tried to implement the the code Thread Safe as mentioned, But Ive done as per my knowledge using Synchonized method to make it Thread Safe but could not implement the dead lock logic.

Thanks !

Sayali B Kale.

sayalikaleofficial@gmail.com

GitHub UserID : Sayali-Kale-Official