Rigorous Software Development
Bank account

Explanation of how each of the requirements is addressed in the model

Lampros Barmpounis
Stefan Vikoler
Gautier Candaes

1. **The bank deals with clients and accounts.**

CONTEXT
bankaccount_C0 ›
SETS
- CLIENTS ›
- ACCOUNTS ›

AXIOMS
- axm1: finite(CLIENTS) not theorem ›
- axm2: finite(ACCOUNTS) not theorem ›

END

We define a set of clients and accounts that are finished. The size is defined by the number of clients and accounts.

2. **Opening an account permanently associates it with a single client, the owner of the account.**

- open: not extended ordinary ›

ANY
- cli ›
- acc ›

To open an account we need a client, and an account.

WHERE
- isClient: cli ∈ CLIENTS not theorem ›
- noRepeat: acc ∈ ACCOUNTS∖account not theorem ›

We must make sure that the client is a real client from the bank, and that the account doesn't exist yet, in other words the uniqueness of the account.

THEN
- setBal: balance(acc) ≔ 0 ›
- setCli: client(acc) ≔ cli ›

END

We initialize the account balance to 0 and associate the client to this account.

3. **Every account has a balance.**
4. **The balance of any newly opened account is zero.**

By this piece of code, (extract of the open method)

setBal: balance(acc) ≔ 0 ›

we can say that every account has a balance which is initialized to 0.

5. **The balance of an account cannot be negative at any moment.**

INVARIANTS

- invAcc:account ⊆ ACCOUNTS not theorem ›
- invBal: balance ∈ account → ℕ1 theorem ›
- invCli: client ∈ account → CLIENTS not theorem ›

As we put the balance belong to the Natural strictly greater than zero, this invariant assure that the balance of an account can't be negative at any moment.
Moreover, for the withdraw and transfer operations, we make the necessary verifications :

- isEnough: balance(acc) ≥ amount theorem ›

We make sure before doing the operation that the account has at least the amount necessary.

6. **Bank clients can deposit money in any existing account, thereby increasing its balance.**

- deposit: not extended ordinary ›

ANY

- acc ›
- amount ›
- cli ›

To make this operation we need these variables because to deposit, a client need an account and a amount to put in it.

WHERE

- isCli: cli ∈ CLIENTS not theorem ›
- isAcc: acc ∈ account not theorem ›
- isPos: amount ≥ 0 not theorem ›

We make sure that the client is a client from the bank, he has an account and the amount he want to deposit is positive.

THEN

- incBal: balance(acc) ≔ balance(acc) + amount ›

END

We add the amount to the balance of the account concerned.

7. **Money can be withdrawn from an existing account, and its balance is decreased by that amount.**

- withdraw: not extended ordinary ›

ANY

- acc ›
- amount ›
- cli ›

To withdraw money, a client need to have an account and indicate an amount.

WHERE

- isAcc: acc ∈ account not theorem ›
- isCli: cli ∈ CLIENTS not theorem ›
- valCli: client(acc) = cli not theorem ›
- isPos: amount ≥ 0 not theorem ›
- isEnough: balance(acc) ≥ amount theorem ›

We make sure that the person is a client, he has an account and it is the owner of this account. Finally we check that the amount is positive and he has enough money on his account to withdraw this amount.

THEN

- decBal: balance(acc) ≔ balance(acc) − amount ›

END

We decrease the amount of his account balance.

8. **Money can be transferred from between two existing accounts. The balance of the account from which money is transferred is reduced by the amount being transferred, and the balance of the account to which money is transferred is increased by the amount being transferred.**

- transfer: not extended ordinary ›

ANY

- fromAcc ›
- toAcc ›
- amount ›
- cli ›

To transfer money, a client need indicate an amount to transfer from an account to another account.

WHERE

- isAcc1:fromAcc ∈ account not theorem ›
- isAcc2:toAcc ∈ account not theorem ›
- isPos: amount ≥ 0 not theorem ›
- isCli: cli ∈ CLIENTS not theorem ›
- valCli: client(fromAcc) = cli not theorem ›
- isEnough: balance(fromAcc) ≥ amount not theorem ›

We make sure the two accounts exist, that the person is the client and the owner of the source account and he has enough money on his account to transfer it.

THEN

- setBal: balance ≔ ({fromAcc, toAcc} ⩤ balance) ∪ {fromAcc ↦ balance(fromAcc) − amount} ∪ {toAcc ↦ balance(toAcc) + amount} ›

END

We perform the modification on the two accounts, we decrease the source account of the amount and increase the beneficiary account .

9. **Only the owner of an account can withdraw or transfer money from it.**

This piece of code can assure that only the owner of an account can withdraw or transfer money from it :

```
isCli:   cli ∈ CLIENTS not theorem ›
valCli:  client(acc) = cli not theorem ›
```

10. **The decrease of the balance of any account caused by two consecutive operations has to be less than or equal to K (for some constant K).**