<<enum>> Account AccountService AccountType - number: Long accounts: Map<Long, Account> - balance: BigDecimal CREDIT_ACCOUNT, - owner: Owner SAVING_ACCOUNT - accountType: AccountType + deposit(BigDecimal amount, Long account): BigDecimal; + withdraw(BigDecimal amount, Long account): BigDecimal; + getBalance(Long accountNumber): BigDecimal; + transfer(Long sourceAccount, BigDecimal amout, Long targetAccount): BigDecimal + applyIncome() V1 Owner - cpf: String - name: String - interface transparent to user, deposit, withdraw, transfer does not need to know about account specs - one acess to accounts data (all in one place) - apply income must iterate through all accounts, only one data structure - code complexity (conditionals checking account types) - coupling, change in creditAccount withdraw may affect saving Account AccountService number: Long - creditAccounts: Map<Long, Account> - balance: BigDecimal - savingAccounts: Map<Long, Account> owner: Owner + deposit(BigDecimal amount, Long account): BigDecimal; + withdraw(BigDecimal amount, Long account): BigDecimal; + getBalance(Long accountNumber): BigDecimal; + transfer(Long sourceAccount, BigDecimal amout, Long targetAccount): BigDecimal + applyIncome() Owner V2 - cpf: String - name: String - interface transparent to user, deposit, withdraw, transfer does not need to know about account specs - apply income just need to iterate through desired objects - all operations must look into two data structure to find desired - code complexity (conditionals checking account types) - coupling, change in creditAccount withdraw may affect saving account CreditAccountService Account - accounts: Map<Long, Account> number: Long - balance: BigDecimal - owner: Owner + deposit(BigDecimal amount, Long account): BigDecimal; + withdraw(BigDecimal amount, Long account): BigDecimal; + getBalance(Long accountNumber): BigDecimal; + transfer(Long sourceAccount, BigDecimal amout, Long targetAccount): BigDecimal + save(Long sourceAccount, BigDecimal amout, Long saveAccount): BigDecimal SaveAccountService Owner - accounts: Map<Long, Account> cpf: String - name: String + deposit(BigDecimal amount, Long account): BigDecimal;

- + withdraw(BigDecimal amount, Long account): BigDecimal;
- + getBalance(Long accountNumber): BigDecimal;
- + transferForCreditAccount(Long sourceAccount, BigDecimal amout, Long targetAccount): BigDecimal

V3

- simple methods doing one thing
- less coupling, save accounts rules can change without affecting
- one access to data structure in all operations
- apply income just need to iterate through desired objects

- user must know data type (save/credit account) of account he is using