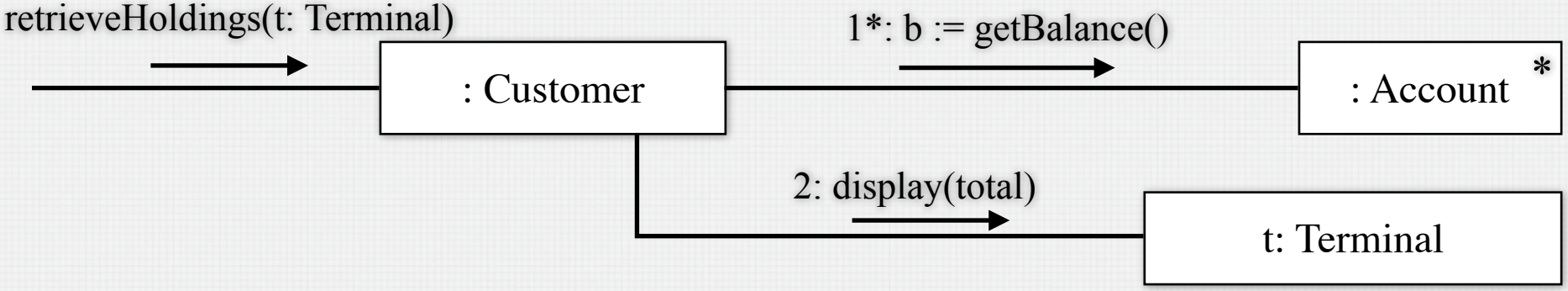
RetrieveHoldings

Message declaration:

Balance (amount: Money);

Operation: Bank::retrieveHoldings(cust: Customer);
Description: Results in the sum of the balances of all the accounts for a given customer being sent to the teller;
Scope: Customer; Owns; Account;
Messages: Teller::{Balance;};
Aliases: total: Money = cust.account.balance→**sum**();
Pre: **true**;
Post: **sender**^balance(total);

RetrieveHoldings Design



operation Customer::retrieveHoldings(id:Terminal)
    foreach account of the customer
        get the balance (1);
        add it to the total holdings;
    end foreach;
    display the total holdings on the teller's terminal (2);

Verify (1)

Message declaration:

Warning(acc: String, balance: Money);

Operation: Bank::verify();

Description: Monthly verification of all bank accounts. A warning message is sent to the client if the balance is negative. One message is sent for each account.

Scope: Owns; Account; Customer; Represents;

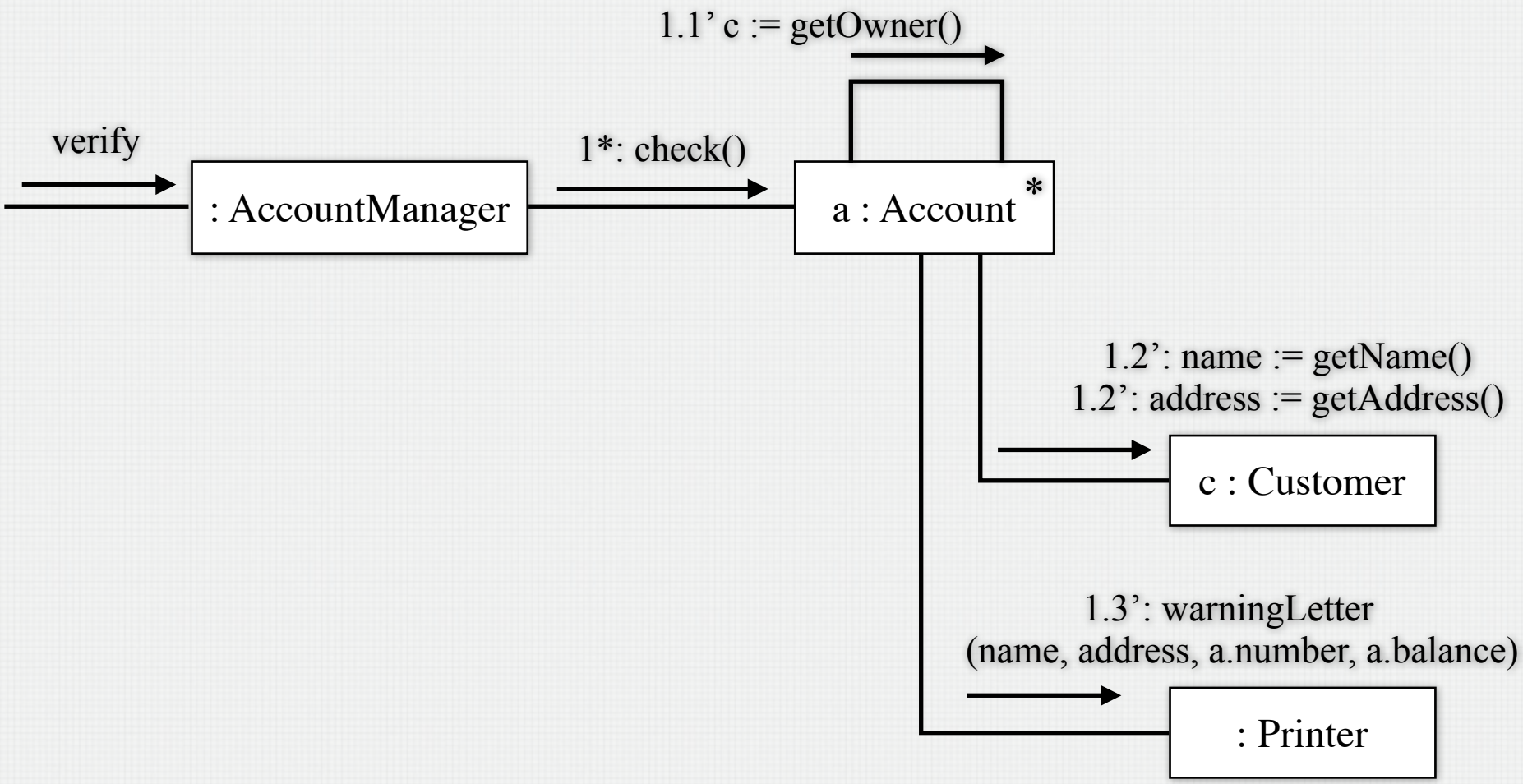
Messages: Client::{Warning;};

Aliases: overdrawnAccounts: Set(Account) =
self.account→**select**(a | a.balance < 0);

Pre: **true**;

Post: overdrawnAccounts→**forall**
(a | a.customer.client^warning(a.number, a.balance));

Verify Design Collaboration Diagram



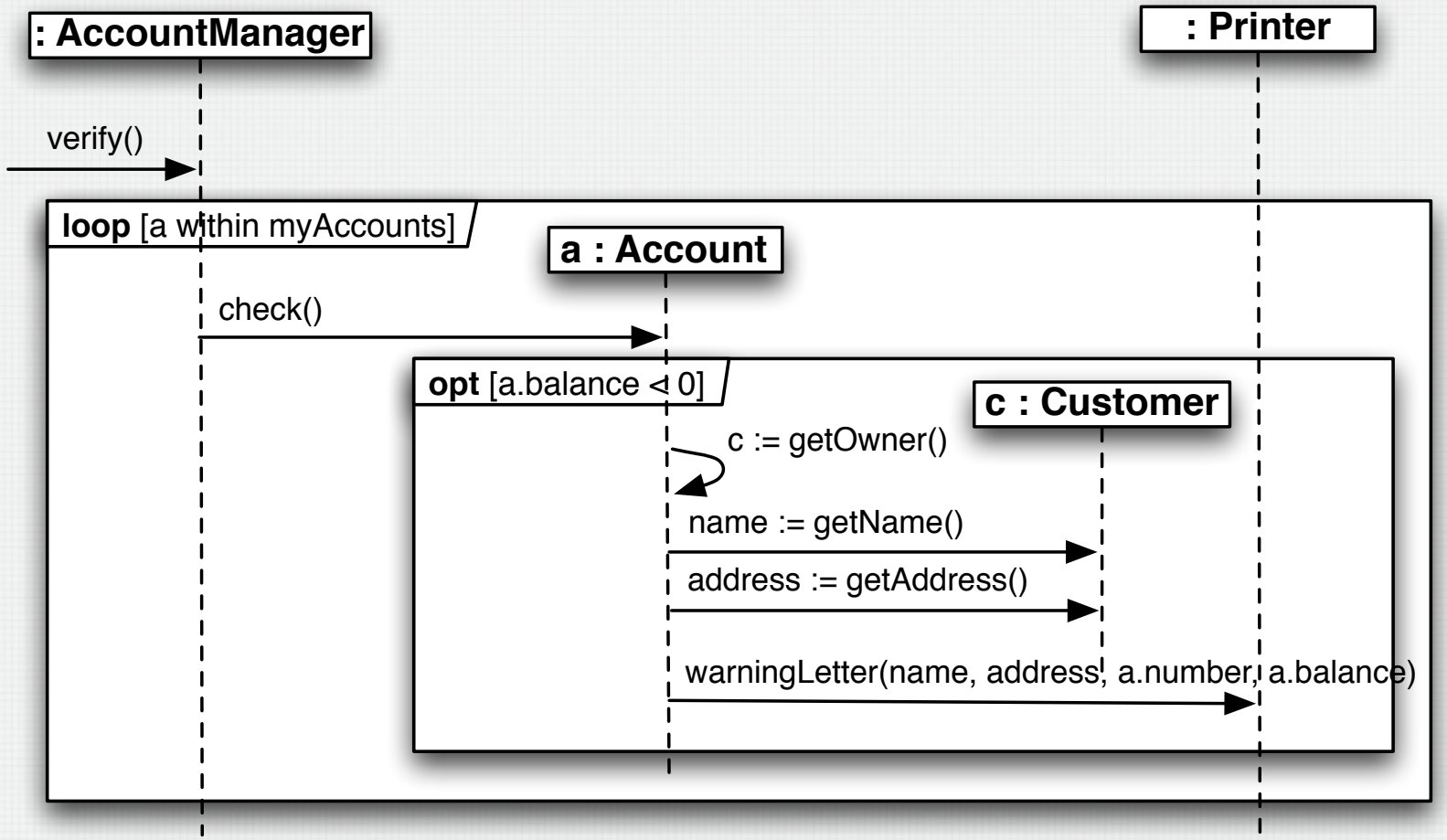
Verify (2)

Message declaration:

type AccountInfo **is TupleType** {acc: String, balance: Money}
Warning(accounts: Set(AccountInfo));

Operation: Bank::verify();
Description: Monthly verification of all bank accounts. A warning message is sent to the client if the balance is negative. One message is sent for each customer.
Scope: Owns; Account; Customer; Represents;
Messages: Client::{Warning;};
Aliases: customersInTrouble: Set(Customer) =
self.customer→**select**(c | c.account→**exists**(a | a.balance < 0));
Pre: **true**;
Post: customersInTrouble→**forall**(c : Customer |
let content: Set(AccountInfo) **in**
c.account→**select**(a | a.balance < 0))→**forAll**(a : Account |
content→**includes**(Tuple{acc = a.number, balance = a.balance})) &
c^warning(content)
end let)

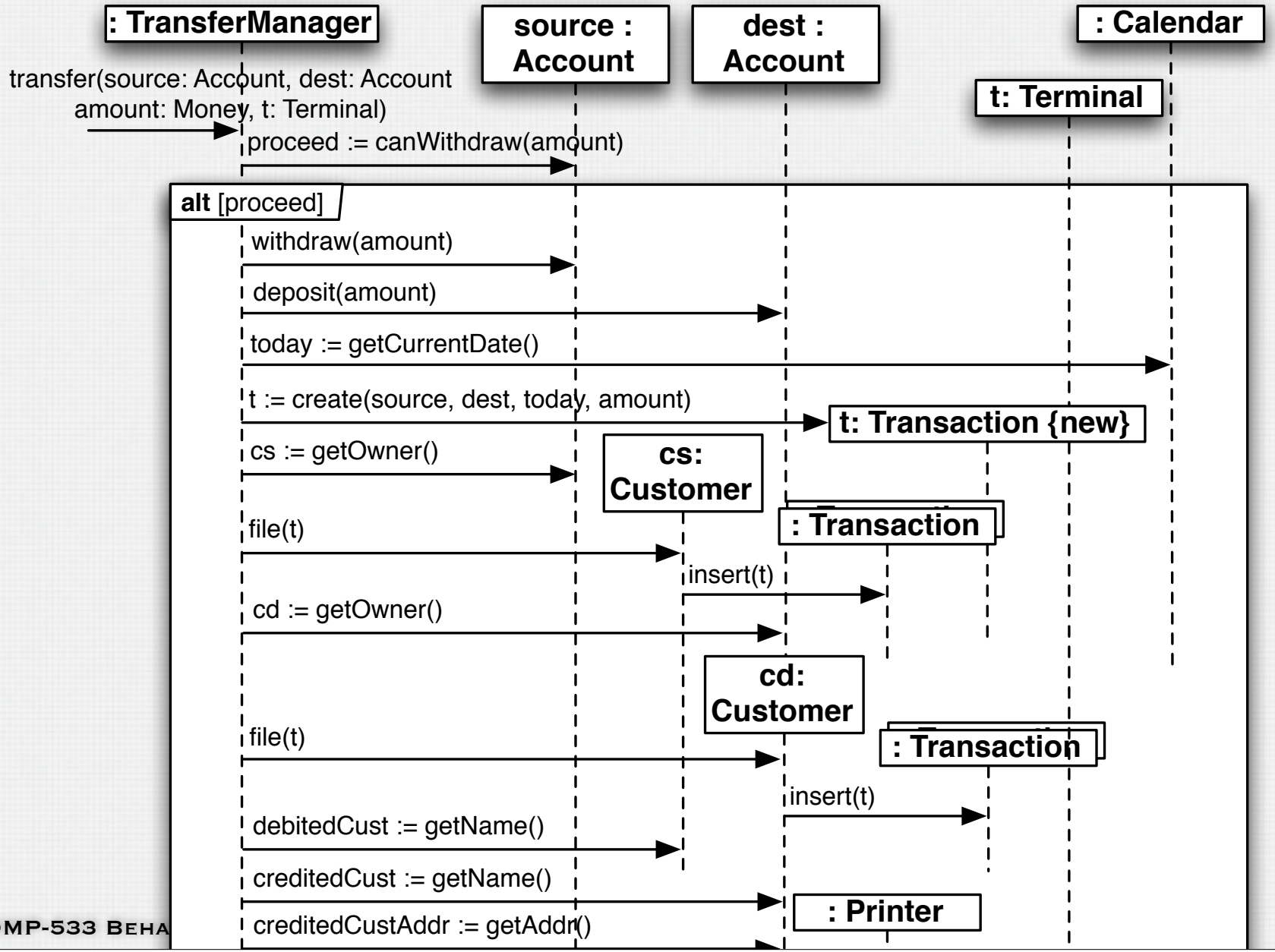
Verify Design Sequence Diagram



Transfer Design Decisions

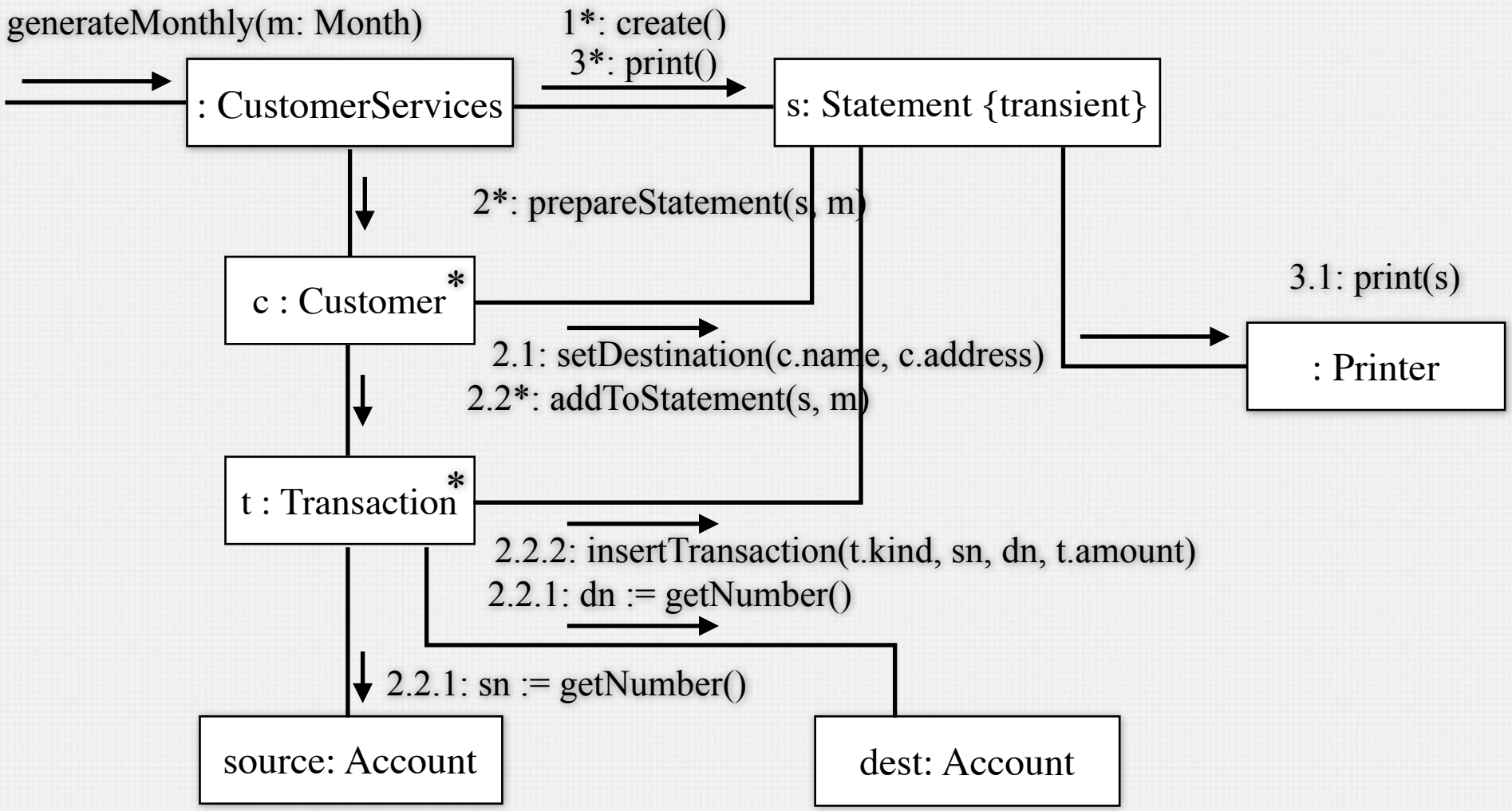
- The TransferManager system-wide object is the controller for all transfer requests
- Communication with the teller is done through a terminal
- Communication with the customer that owns the destination account of the transfer is done by printing a letter
- The TransferManager interacts with the printer
- The customer keeps a record of all transactions involving any of her accounts

Transfer Design



GenerateMonthly Design Decisions

- The CustomerServices system-wide object is the controller for GenerateMonthlyReports
  - CustomerServices knows about all customers in the system
- The Customer and Transaction class should not be dependent on the Printer class
  - We introduce an intermediate Statement class
- The Customer and Transaction class are responsible for sending the state that they are encapsulating to the statement


generateMonthly(m: Month)
: CustomerServices
1*: create()
3*: print()
s: Statement {transient}
2*: prepareStatement(s, m)
c : Customer*
3.1: print(s)
: Printer
2.1: setDestination(c.name, c.address)
2.2*: addToStatement(s, m)
t : Transaction*
2.2.2: insertTransaction(t.kind, sn, dn, t.amount)
2.2.1: dn := getNumber()
2.2.1: sn := getNumber()
source: Account
dest: Account