EXAMPLE: BANKING APPLICATION

The following concepts all represent classes, which are related according to the problem statement.

Bank	Deposit	Account	Saving
Customer	Transfer	CreditCard	Checking
Withdrawal	Transaction	Passbook	

Banks issue credit cards (e.g., Visa, MasterCard, etc.) to customers. Each credit card is issued by only one bank to only one customer. Customers hold accounts with banks. Each account is with only one bank and held by only one customer. An account may be a saving account or a checking account. Each saving account has a passbook, which shows the transactions made against the account. Checking accounts do not have passbooks. Customers make transactions against their accounts. A transaction can be a deposit, withdrawal or transfer.

Construct a class diagram, using *only* the eleven given classes.

EXAMPLE: BANKING APPLICATION—SOLUTION

Analysis of Problem Statement

Since we are only required to show how the classes are related, our analysis will focus only on discovering the associations, aggregations, compositions and generalizations among the classes and their related multiplicities.

Banks issue credit cards (e.g., Visa, MasterCard, etc.) to customers.

associations: Bank *Issues* CreditCard
CreditCard *IsFor* Customer

Each credit card is issued by only one bank to only one customer.

multiplicities: min-card(CreditCard, *Issues*) = 1
max-card(CreditCard, *Issues*) = 1
min-card(CreditCard, *IsFor*) = 1
max-card(CreditCard, *IsFor*) = 1

EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)

Customers hold accounts with banks.

associations: Customer *Holds* Account
Account *IsWith* Bank

Each account is with only one bank and held by only one customer.

multiplicities: min-card(Account, *IsWith*) = 1
max-card(Account, *IsWith*) = 1
min-card(Account, *Holds*) = 1
max-card(Account, *Holds*) = 1

An account may be a saving account or a checking account.

generalizations: Account *generalizes* Saving
Account *generalizes* Checking

EXAMPLE: BANKING APPLICATION—SOLUTION (cont'd)

Each saving account has a passbook, which shows the transactions made against the account.

associations: Saving *Has* Passbook

multiplicities: min-card(Saving, *Has*) = 1

Remarks: Each saving account has at least one passbook.

Checking accounts do not have passbooks.

Information about checking accounts; nothing to represent.

Customers make transactions against their accounts.

associations: Customer *Makes* Transaction

Transaction *IsAgainst* Account

EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)

A transaction can be a deposit, withdrawal or transfer.

generalizations: Transaction *generalizes* Deposit
Transaction *generalizes* Withdrawal
Transaction *generalizes* Transfer

EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)

Real World Knowledge

From our real world knowledge, we can reasonably infer the following multiplicities:

A bank can issue zero or many credit cards.

multiplicities: min-card(Bank, *Issues*) = 0
max-card(Bank, *Issues*) = *

A customer can have zero or many credit cards.

multiplicities: min-card(Customer, *IsFor*) = 0
max-card(Customer, *IsFor*) = *

A customer may hold no account (they may be a credit card holder only) and can hold many.

multiplicities: min-card(Customer, *Holds*) = 0
max-card(Customer, *Holds*) = *

EXAMPLE: BANKING APPLICATION—SOLUTION (CONT'D)

A bank can have many accounts; whether it needs to have at least one is not known.

multiplicities: min-card(Bank, *IsWith*) = 0 or 1
max-card(Bank, *IsWith*) = *

A customer can make zero or many transactions.

multiplicities: min-card(Customer, *Makes*) = 0
max-card(Customer, *Makes*) = *

Every transaction is made by exactly one customer.

multiplicities: min-card(Transaction, *Makes*) = 1
max-card(Transaction, *Makes*) = 1

A transaction is against at least one and at most two accounts (if it is a transfer).

multiplicities: min-card(Transaction, *IsAgainst*) = 1
max-card(Transaction, *IsAgainst*) = 2

EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)

An account can have zero or many transactions made against it.

multiplicities: min-card(Account, *IsAgainst*) = 0
max-card(Account, *IsAgainst*) = *

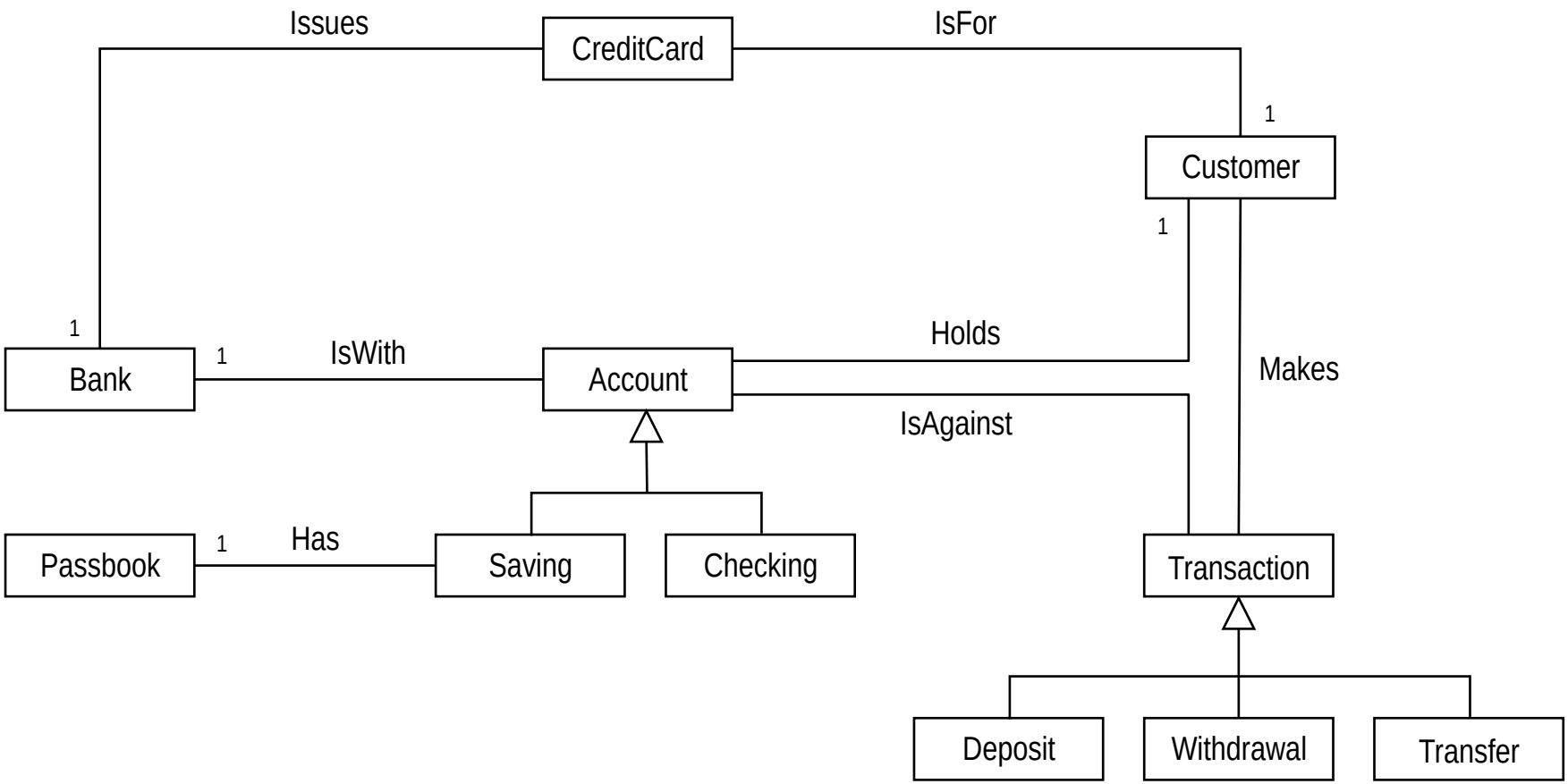
Every saving account has at most one passbook.

multiplicities: max-card(Saving, *Has*) = 1

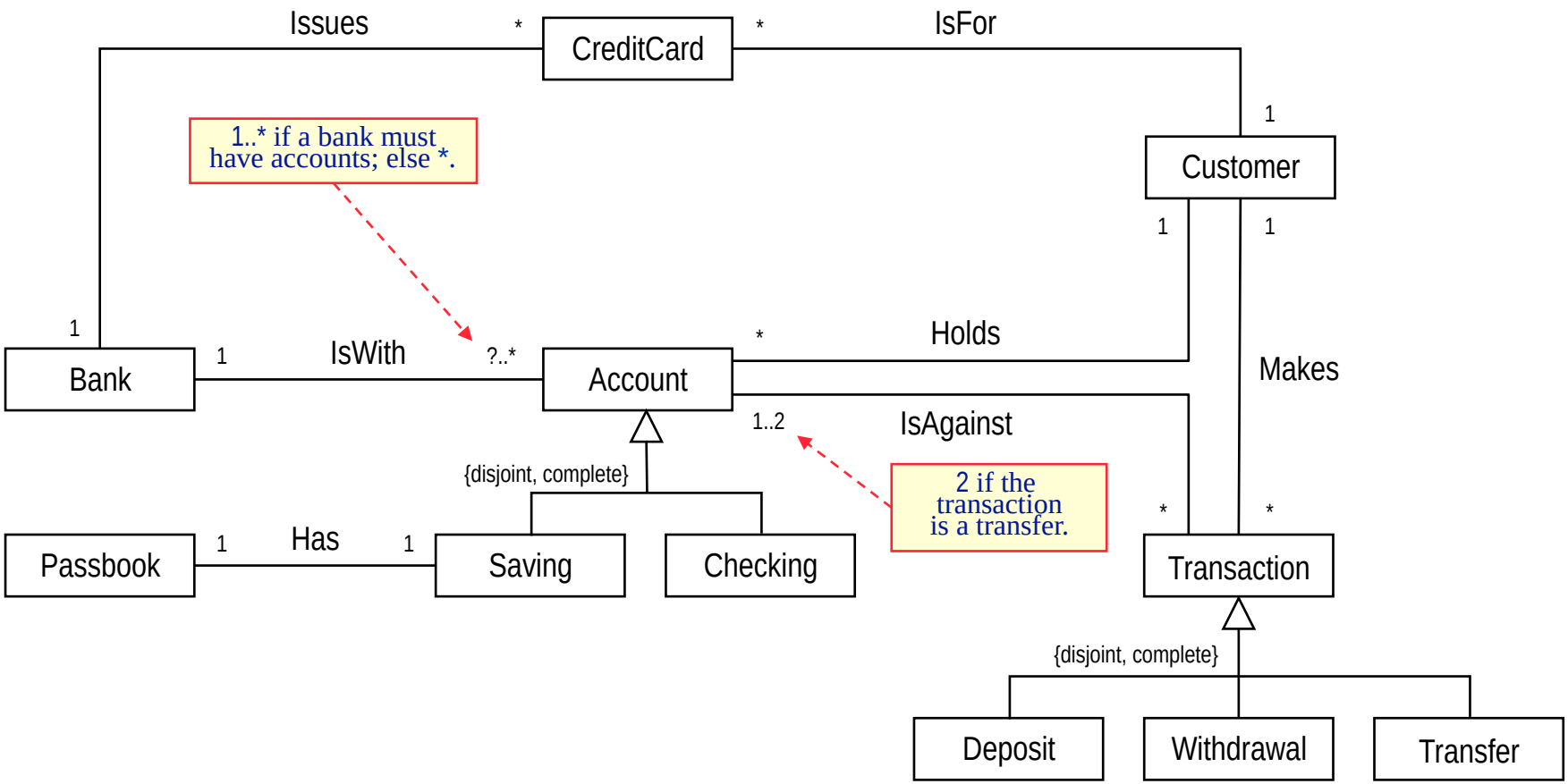
Every passbook is for exactly one saving account.

multiplicities: min-card(Passbook, *Has*) = 1
max-card(Passbook, *Has*) = 1

EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)



EXAMPLE: BANKING APPLICATION—SOLUTION
(cont'd)



EXAMPLE: BANKING APPLICATION — COMMON ERRORS

- Missing/incorrect associations
- Missing association names
  - While associations names are not required, it is always a good idea to name them.
- Incorrect use of aggregation
  - There is no aggregation in this example!
- Incorrect use of generalization
  - Ask “kind of” question.
- Including operations as associations
  - Associations do not represent operations, but result of operations.
- Using modeling elements incorrectly
  - (e.g., xor)