Chapter 25
ATM Case Study, Part 1: Object-Oriented Design with the UML

C++ How to Program, 9/e

OBJECTIVES

In this chapter you'll:

- Learn a simple object-oriented design methodology.
- Learn what a requirements document is.
- Identify classes and class attributes from a requirements document.
- Identify objects' states, activities and operations from a requirements document.
- Determine the collaborations among objects in a system.
- Work with the UML's use case, class, state, activity, communication and sequence diagrams to graphically model a simple object-oriented system.

25.1 Introduction
25.2 Introduction to Object-Oriented Analysis and Design
25.3 Examining the ATM Requirements Document
25.4 Identifying the Classes in the ATM Requirements Document
25.5 Identifying Class Attributes
25.6 Identifying Objects' States and Activities
25.7 Identifying Class Operations
25.8 Indicating Collaboration Among Objects
25.9 Wrap-Up

25.3 Examining the ATM Requirements Document

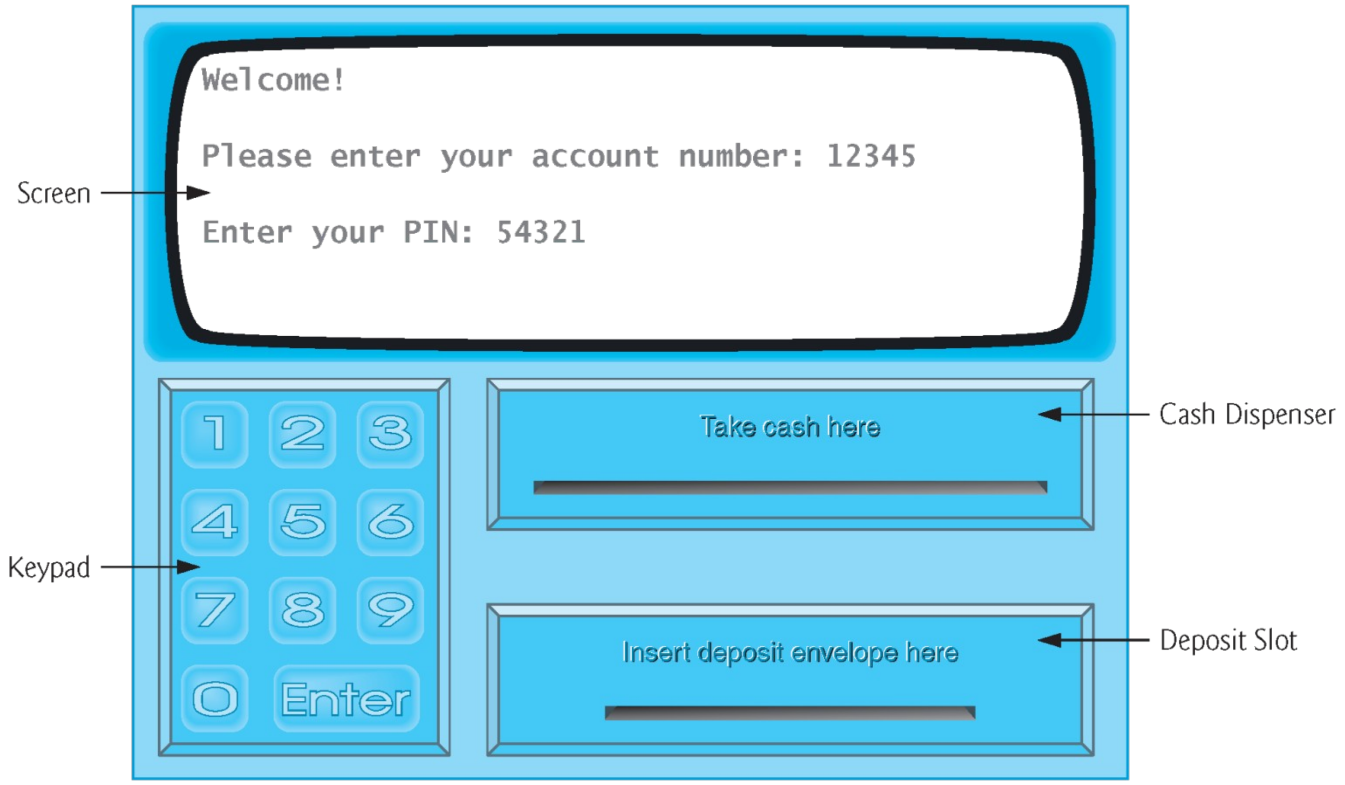
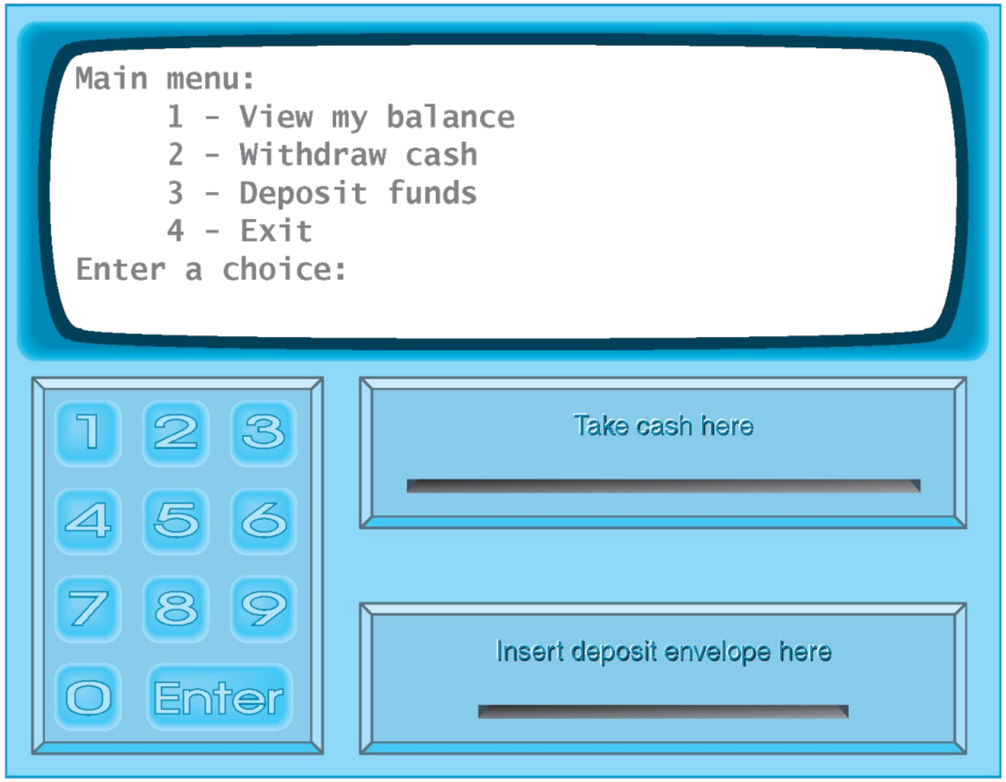
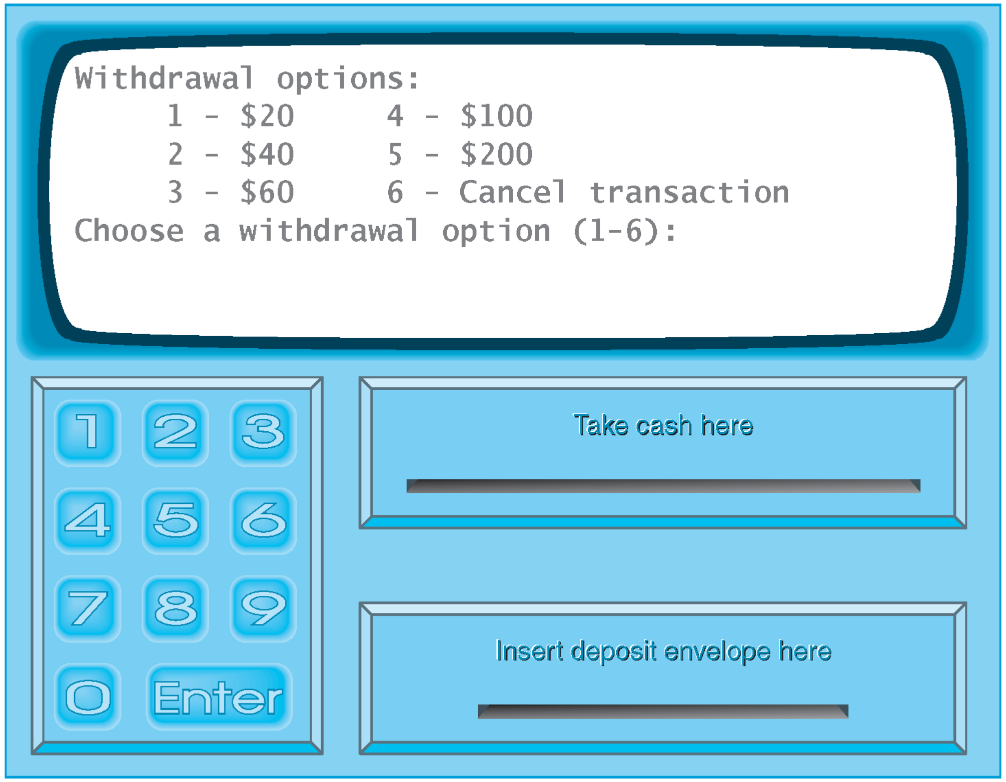

```
Welcome!

Please enter your account number: 12345

Enter your PIN: 54321
```

Fig. 25.1 | Automated teller machine user interface.



```
Main menu:
     1 - View my balance
     2 - Withdraw cash
     3 - Deposit funds
     4 - Exit
Enter a choice:
```

Fig. 25.2 | ATM main menu.



```
Withdrawal options:
     1 - $20      4 - $100
     2 - $40      5 - $200
     3 - $60      6 - Cancel transaction
Choose a withdrawal option (1-6):
```

Fig. 25.3 | ATM withdrawal menu.

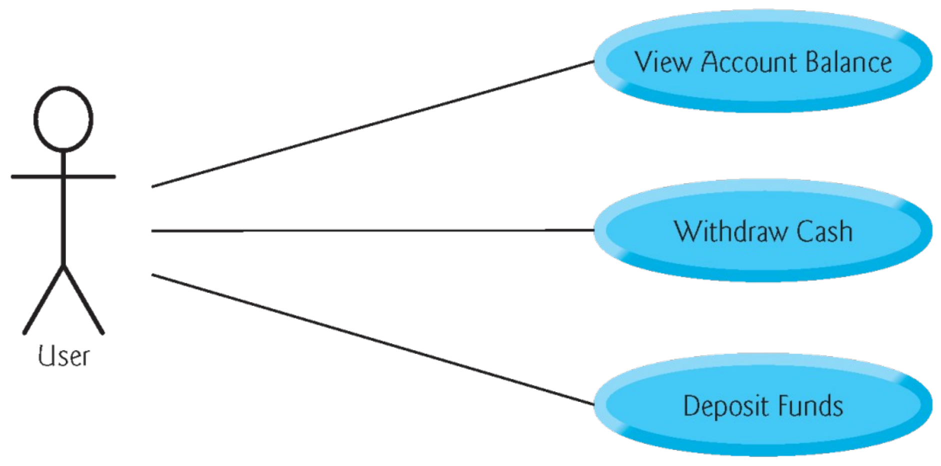


Fig. 25.4 | Use case diagram for the ATM system from the User's perspective.

25.4 Identifying the Classes in the ATM Requirements Document

Nouns and noun phrases in the requirements document			
bank	money / fund	account number	ATM
screen	PIN	user	keypad
bank database	customer	cash dispenser	balance inquiry
transaction	$20 bill / cash	withdrawal	account
deposit slot	deposit	balance	deposit envelope

Fig. 25.5 | Nouns and noun phrases in the requirements document.

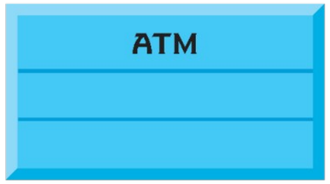


Fig. 25.6 | Representing a class in the UML using a class diagram.

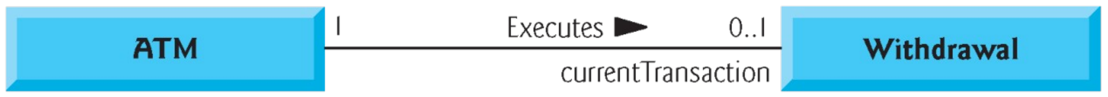


Fig. 25.7 | Class diagram showing an association among classes.

Symbol	Meaning
0	None
1	One
m	An integer value
0..1	Zero or one
m, *n*	*m* or *n*
m..*n*	At least *m*, but not more than *n*
*	Any nonnegative integer (zero or more)
0..*	Zero or more (identical to *)
1..*	One or more

Fig. 25.8 | Multiplicity types.

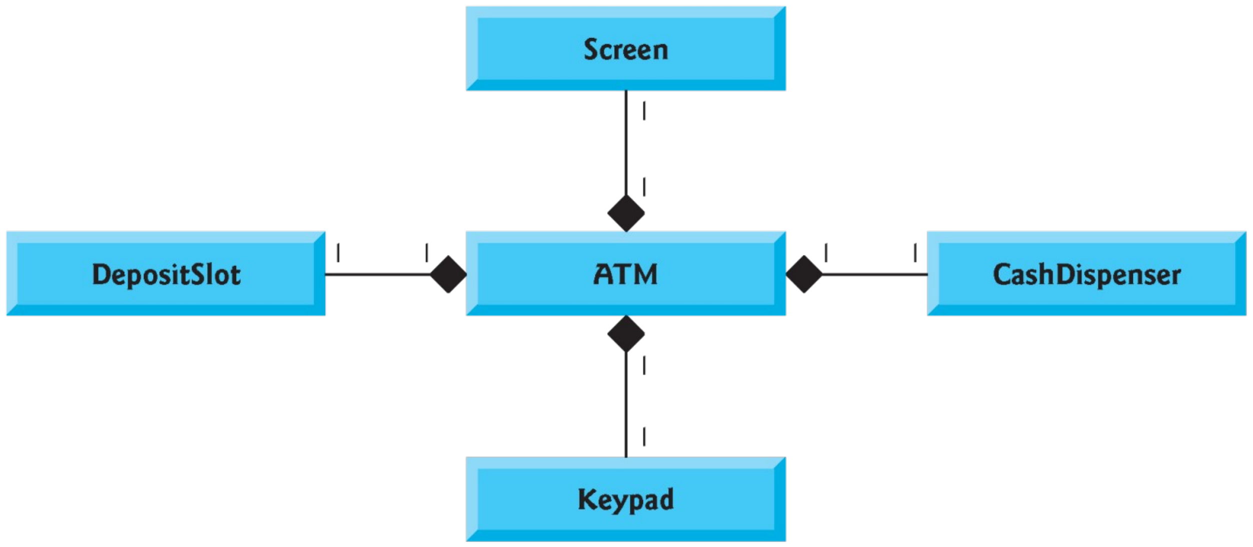


Fig. 25.9 | Class diagram showing composition relationships.

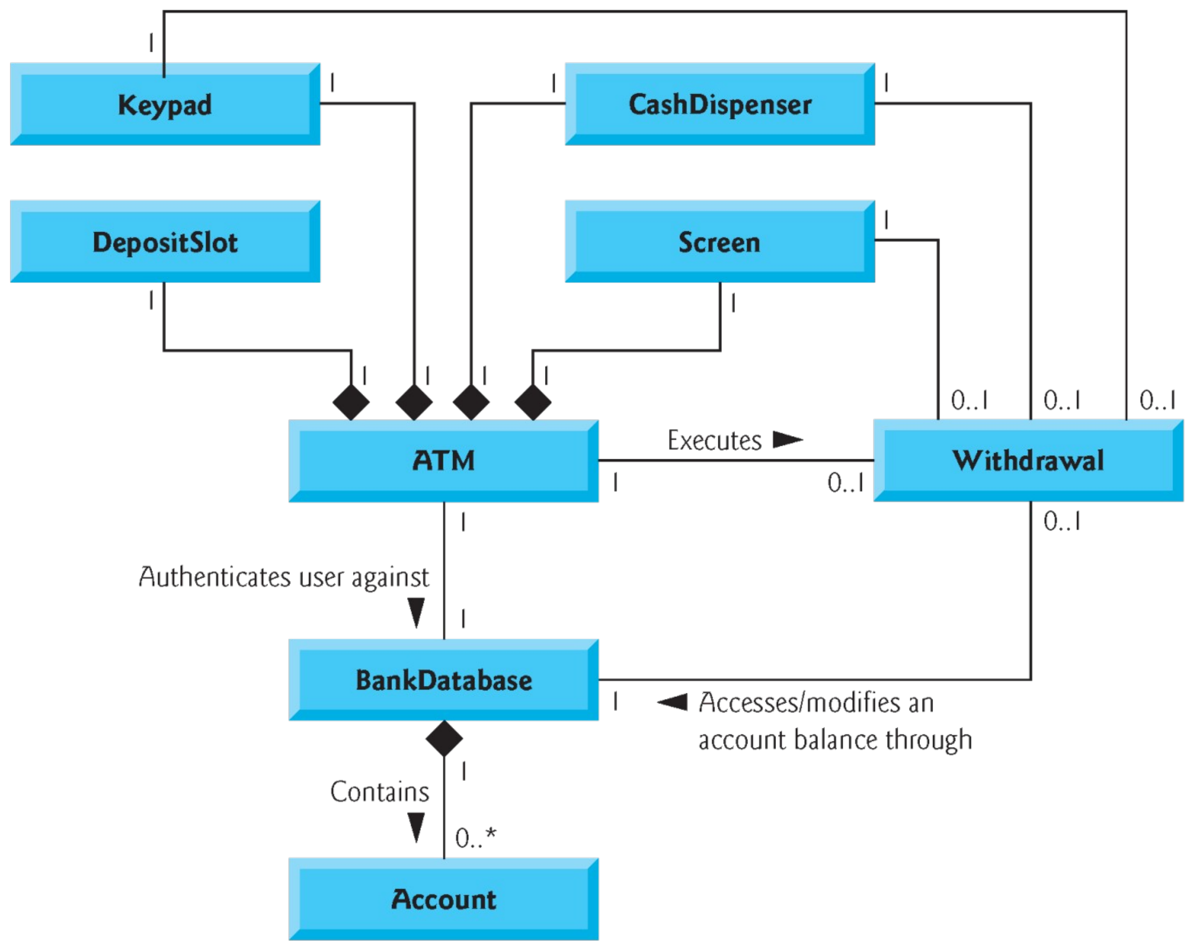


Fig. 25.10 | Class diagram for the ATM system model.

25.5 Identifying Class Attributes

Class	Descriptive words and phrases
`ATM`	user is authenticated
`BalanceInquiry`	account number
`Withdrawal`	account number amount
`Deposit`	account number amount
`BankDatabase`	[no descriptive words or phrases]
`Account`	account number PIN balance
`Screen`	[no descriptive words or phrases]
`Keypad`	[no descriptive words or phrases]
`CashDispenser`	begins each day loaded with 500 $20 bills
`DepositSlot`	[no descriptive words or phrases]

Fig. 25.11 | Descriptive words and phrases from the ATM requirements.

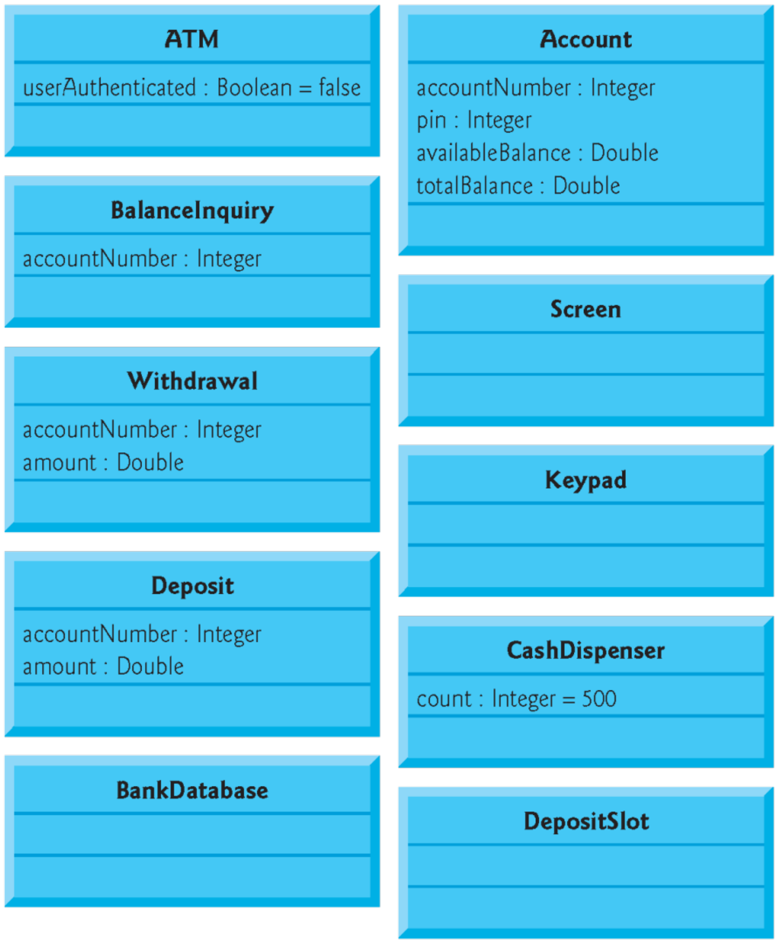


Fig. 25.12 | Classes with attributes.

Software Engineering Observation 25.1

At the early stages in the design process, classes often lack attributes (and operations). Such classes should not be eliminated, however, because attributes (and operations) may become evident in the later phases of design and implementation.

25.6 Identifying Objects' States and Activities

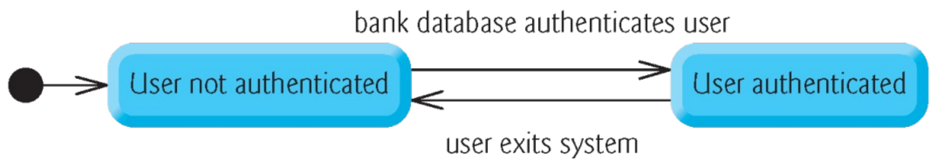
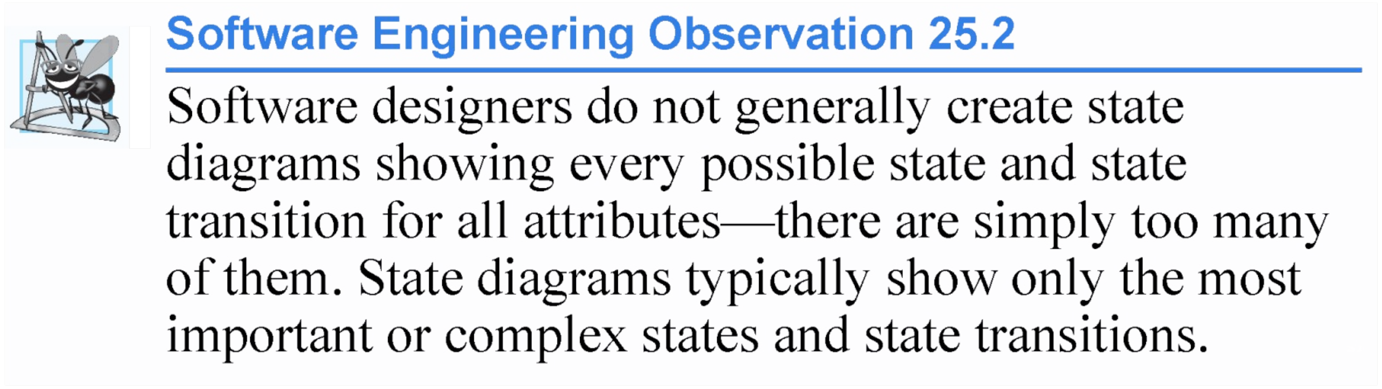


Fig. 25.13 | State diagram for the ATM object.



Software Engineering Observation 25.2

Software designers do not generally create state diagrams showing every possible state and state transition for all attributes—there are simply too many of them. State diagrams typically show only the most important or complex states and state transitions.

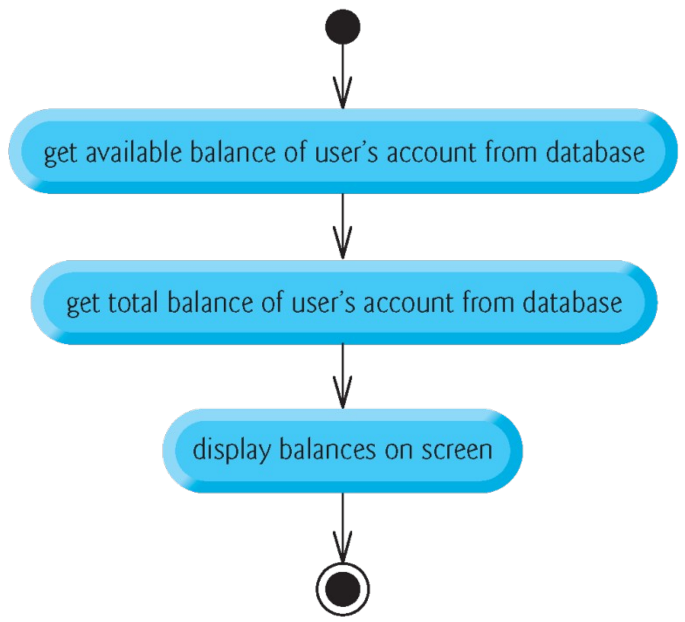

get available balance of user's account from database
get total balance of user's account from database
display balances on screen

Fig. 25.14 | Activity diagram for a `BalanceInquiry` transaction.

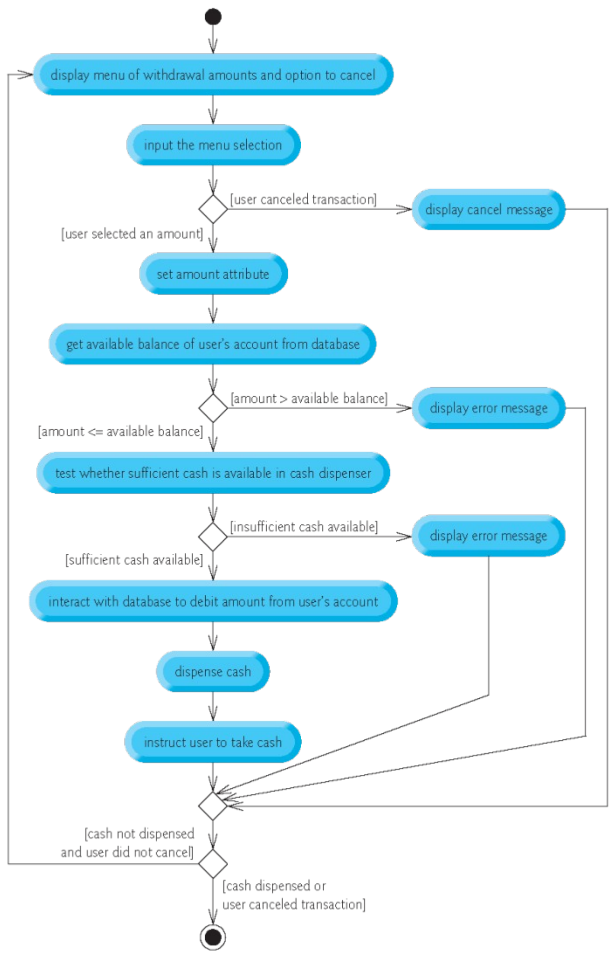

display menu of withdrawal amounts and option to cancel
input the menu selection
[user canceled transaction]
display cancel message
[user selected an amount]
set amount attribute
get available balance of user's account from database
[amount > available balance]
display error message
[amount <= available balance]
test whether sufficient cash is available in cash dispenser
[insufficient cash available]
display error message
[sufficient cash available]
interact with database to debit amount from user's account
dispense cash
instruct user to take cash
[cash not dispensed and user did not cancel]
[cash dispensed or user canceled transaction]

Fig. 25.15 | Activity diagram for a `Withdrawal` transaction.

25.7 Identifying Class Operations

Class	Verbs and verb phrases
`ATM`	executes financial transactions
`BalanceInquiry`	[none in the requirements document]
`Withdrawal`	[none in the requirements document]
`Deposit`	[none in the requirements document]
`BankDatabase`	authenticates a user, retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account
`Account`	retrieves an account balance, credits a deposit amount to an account, debits a withdrawal amount from an account
`Screen`	displays a message to the user
`Keypad`	receives numeric input from the user
`CashDispenser`	dispenses cash, indicates whether it contains enough cash to satisfy a withdrawal request
`DepositSlot`	receives a deposit envelope

Fig. 25.16 | Verbs and verb phrases for each class in the ATM system.

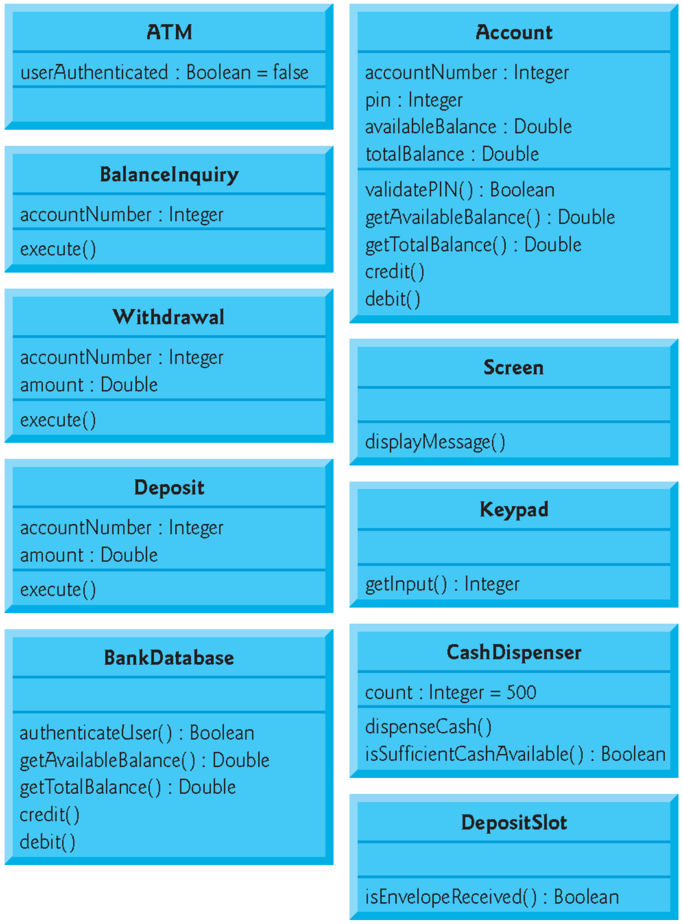


Fig. 25.17 | Classes in the ATM system with attributes and operations.

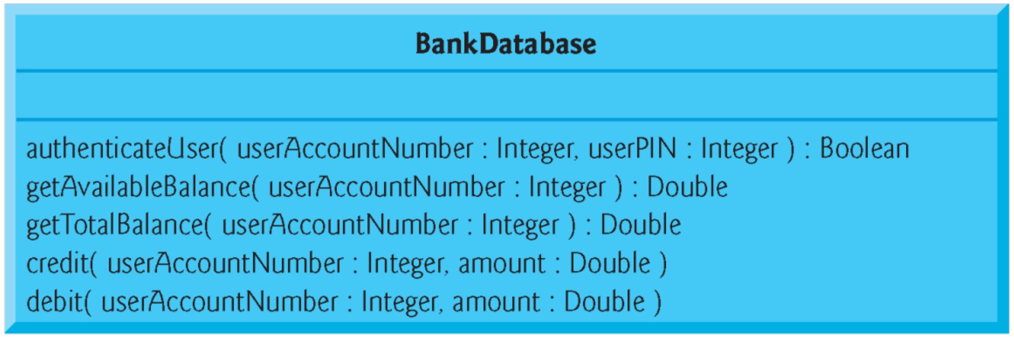

BankDatabase
authenticateUser(userAccountNumber : Integer, userPIN : Integer) : Boolean
getAvailableBalance(userAccountNumber : Integer) : Double
getTotalBalance(userAccountNumber : Integer) : Double
credit(userAccountNumber : Integer, amount : Double)
debit(userAccountNumber : Integer, amount : Double)

Fig. 25.18 | Class `BankDatabase` with operation parameters.

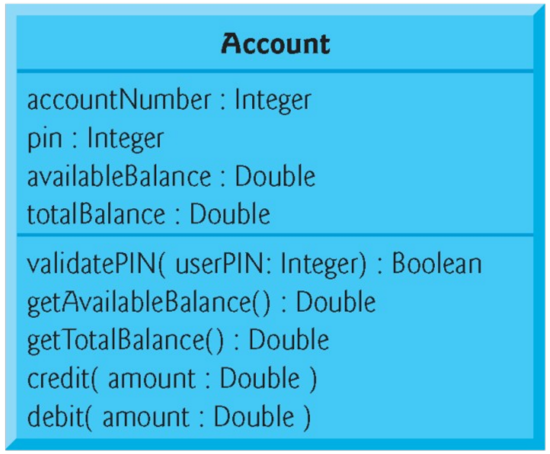

Account
accountNumber : Integer
pin : Integer
availableBalance : Double
totalBalance : Double
validatePIN(userPIN: Integer) : Boolean
getAvailableBalance() : Double
getTotalBalance() : Double
credit(amount : Double)
debit(amount : Double)

Fig. 25.19 | Class `Account` with operation parameters.

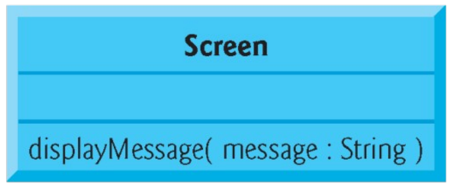


Fig. 25.20 | Class `Screen` with operation parameters.

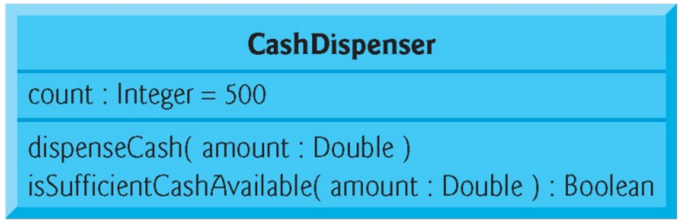

CashDispenser
count : Integer = 500
dispenseCash(amount : Double)
isSufficientCashAvailable(amount : Double) : Boolean

Fig. 25.21 | Class `CashDispenser` with operation parameters.

25.8 Indicating Collaboration Among Objects

An object of class…	sends the message…	to an object of class…
`ATM`	`displayMessage`	`Screen`
	`getInput`	`Keypad`
	`authenticateUser`	`BankDatabase`
	`execute`	`BalanceInquiry`
	`execute`	`Withdrawal`
	`execute`	`Deposit`
`BalanceInquiry`	`getAvailableBalance`	`BankDatabase`
	`getTotalBalance`	`BankDatabase`
	`displayMessage`	`Screen`
`Withdrawal`	`displayMessage`	`Screen`
	`getInput`	`Keypad`
	`getAvailableBalance`	`BankDatabase`
	`isSufficientCashAvailable`	`CashDispenser`
	`debit`	`BankDatabase`
	`dispenseCash`	`CashDispenser`
`Deposit`	`displayMessage`	`Screen`
	`getInput`	`Keypad`
	`isEnvelopeReceived`	`DepositSlot`
	`credit`	`BankDatabase`

Fig. 25.22 | Collaborations in the ATM system. (Part 1 of 2.)

An object of class…	sends the message…	to an object of class…
`BankDatabase`	`validatePIN`	`Account`
	`getAvailableBalance`	`Account`
	`getTotalBalance`	`Account`
	`debit`	`Account`
	`credit`	`Account`

Fig. 25.22 | Collaborations in the ATM system. (Part 2 of 2.)

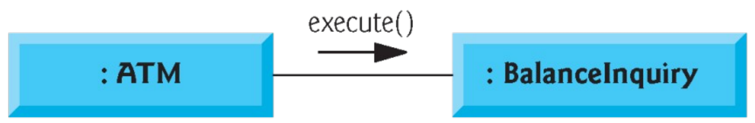


Fig. 25.23 | Communication diagram of the ATM executing a balance inquiry.

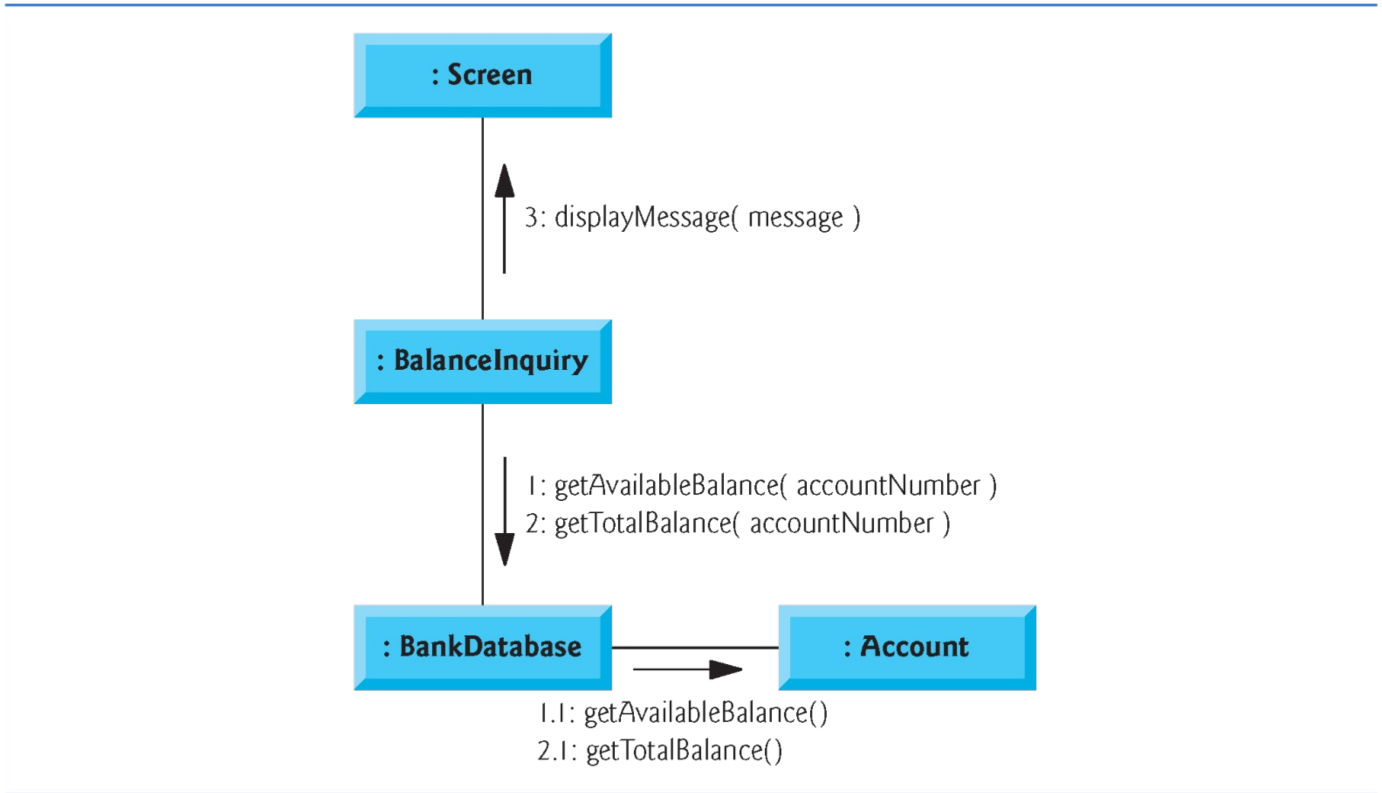

: Screen
3: displayMessage(message)
: BalanceInquiry
1: getAvailableBalance(accountNumber)
2: getTotalBalance(accountNumber)
: BankDatabase
: Account
1.1: getAvailableBalance()
2.1: getTotalBalance()

Fig. 25.24 | Communication diagram for executing a balance inquiry.

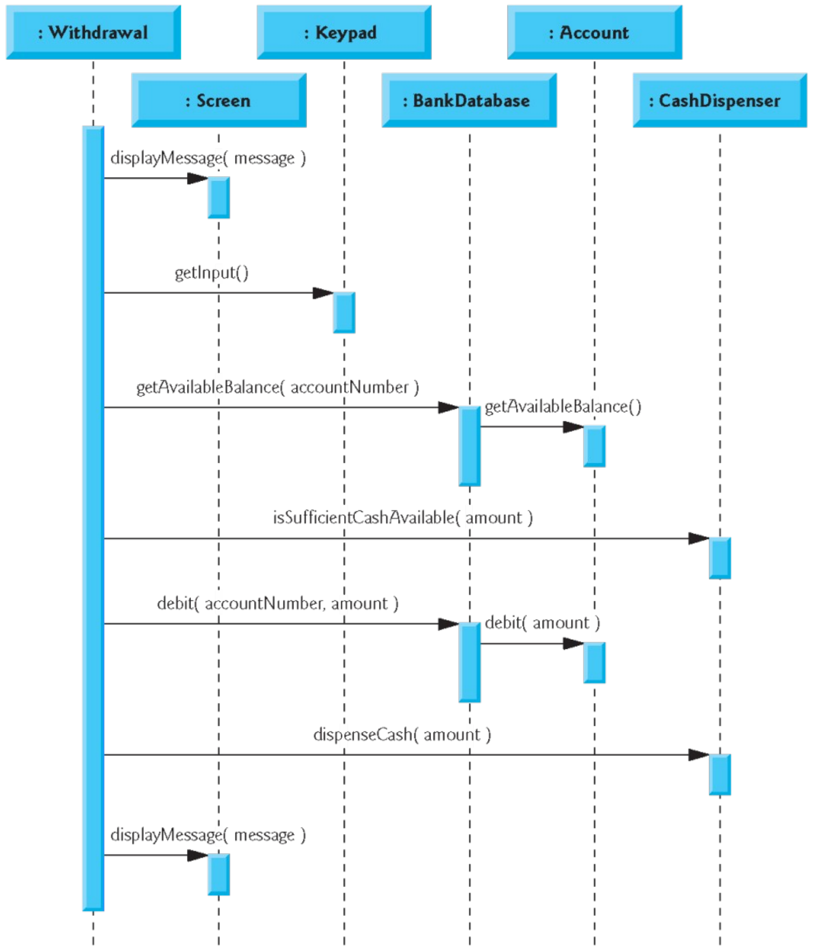

: Withdrawal
: Keypad
: Account
: Screen
: BankDatabase
: CashDispenser
displayMessage(message)
getInput()
getAvailableBalance(accountNumber)
getAvailableBalance()
isSufficientCashAvailable(amount)
debit(accountNumber, amount)
debit(amount)
dispenseCash(amount)
displayMessage(message)

Fig. 25.25 | Sequence diagram that models a `Withdrawal` executing.