3.8 Placing a Class in a Separate File for Reusability

- `.cpp` file is known as a source-code file
- Header files
  - Separate files in which class definitions are placed
    - Allow compiler to recognize the classes when used elsewhere
  - Generally have `.h` filename extensions
- Driver files
  - Program used to test software (such as classes)
  - Contains a `main` function so it can be executed

Outline

```
1  // Fig. 3.9: GradeBook.h
2  // GradeBook class definition in a separate file from main.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include <string> // class GradeBook uses C++ standard string class
8  using std::string;
9
10 // GradeBook class definition
11 class GradeBook
12 {
13 public:
14    // constructor initializes courseName with string supplied as argument
15    GradeBook( string name )
16    {
17       setCourseName( name ); // call set function to initialize courseName
18    } // end GradeBook constructor
19
20    // function to set the course name
21    void setCourseName( string name )
22    {
23       courseName = name; // store the course name in the object
24    } // end function setCourseName
25
```

Class definition is in a header file

fig03_09.cpp

(1 of 2)

Outline

```
26    // function to get the course name
27    string getCourseName()
28    {
29       return courseName; // return object's courseName
30    } // end function getCourseName
31
32    // display a welcome message to the GradeBook user
33    void displayMessage()
34    {
35       // call getCourseName to get the courseName
36       cout << "Welcome to the grade book for\n" << getCourseName()
37          << "!" << endl;
38    } // end function displayMessage
39 private:
40    string courseName; // course name for this GradeBook
41 }; // end class GradeBook
```

fig03_09.cpp

(2 of 2)

Outline

fig03_10.cpp

(1 of 1)

```
1  // Fig. 3.10: fig03_10.cpp
2  // Including class GradeBook from file GradeBook.h for use in main.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "GradeBook.h" // include definition of class GradeBook
8
9  // function main begins program execution
10 int main()
11 {
12    // create two GradeBook objects
13    GradeBook gradeBook1( "CS101 Introduct
14    GradeBook gradeBook2( "CS102 Data Structures in C++" );
15
16    // display initial value of courseName for each GradeBook
17    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
18       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
19       << endl;
20    return 0; // indicate successful termination
21 } // end main
```

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

Including the header file causes the class definition to be copied into the file

3.8 Placing a Class in a Separate File for Reusability (Cont.)

- `#include` preprocessor directive
  - Used to include header files
    - Instructs C++ preprocessor to replace directive with a copy of the contents of the specified file
  - Quotes indicate user-defined header files
    - Preprocessor first looks in current directory
      - If the file is not found, looks in C++ Standard Library directory
  - Angle brackets indicate C++ Standard Library
    - Preprocessor looks only in C++ Standard Library directory

3.8 Placing a Class in a Separate File for Reusability (Cont.)

- **Creating objects**
  - **Compiler must know size of object**
    - **C++ objects typically contain only data members**
    - **Compiler creates one copy of class's member functions**
      - **This copy is shared among all the class's objects**

Error-Prevention Tip 3.3

To ensure that the preprocessor can locate header files correctly, `#include` preprocessor directives should place the names of user-defined header files in quotes (e.g., `"GradeBook.h"`) and place the names of C++ Standard Library header files in angle brackets (e.g., `<iostream>`).

3.9 Separating Interface from Implementation

- **Interface**
  - **Describes what services a class's clients can use and how to request those services**
    - **But does not reveal how the class carries out the services**
    - **A class definition that lists only member function names, return types and parameter types**
      - **Function prototypes**
  - **A class's interface consists of the class's `public` member functions (services)**
- **Separating interface from implementation**
  - **Client code should not break if implementation changes, as long as interface stays the same**

3.9 Separating Interface from Implementation (Cont.)

- **Separating interface from implementation (Cont.)**
  - **Define member functions outside the class definition, in a separate source-code file**
    - **In source-code file for a class**
      - **Use binary scope resolution operator (`::`) to tie each member function to the class definition**
    - **Implementation details are hidden**
      - **Client code does not need to know the implementation**
  - **In header file for a class**
    - **Function prototypes describe the class's `public` interface**

Outline

fig03_11.cpp

(1 of 1)

```
1  // Fig. 3.11: GradeBook.h
2  // GradeBook class definition. This file presents GradeBook's public
3  // interface without revealing the implementations of GradeBook's member
4  // functions, which are defined in GradeBook.cpp.
5  #include <string> // class GradeBook uses C++ standard string class
6  using std::string;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    GradeBook( string ); // constructor that initializes courseName
13    void setCourseName( string ); // function that sets the course name
14    string getCourseName(); // function that gets the course name
15    void displayMessage(); // function that displays a welcome message
16 private:
17    string courseName; // course name for this GradeBook
18 }; // end class GradeBook
```

Interface contains data members and member function prototypes

Common Programming Error 3.8

Forgetting the semicolon at the end of a function prototype is a syntax error.

Good Programming Practice 3.7

Although parameter names in function prototypes are optional (they are ignored by the compiler), many programmers use these names for documentation purposes.

Error-Prevention Tip 3.4

Parameter names in a function prototype (which, again, are ignored by the compiler) can be misleading if wrong or confusing names are used. For this reason, many programmers create function prototypes by copying the first line of the corresponding function definitions (when the source code for the functions is available), then appending a semicolon to the end of each prototype.

Common Programming Error 3.9

When defining a class's member functions outside that class, omitting the class name and binary scope resolution operator (::) preceding the function names causes compilation errors.

Outline

```
1  // Fig. 3.12: GradeBook.cpp
2  // GradeBook member-function definitions. This file contains
3  // implementations of the member functions prototyped in GradeBook.h.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argu
11 GradeBook::GradeBook( string name )
12 {
13    setCourseName( name ); // call set function to initialize courseName
14 } // end GradeBook constructor
15
16 // function to set the course name
17 void GradeBook::setCourseName( string name )
18 {
19    courseName = name; // store the course name in the object
20 } // end function setCourseName
21
```

GradeBook implementation is placed in a separate source-code file

...3_12.cpp

(1 of 2)

Include the header file to access the class name **GradeBook**

Binary scope resolution operator ties a function to its class

Outline

```
22 // function to get the course name
23 string GradeBook::getCourseName()
24 {
25    return courseName; // return object's courseName
26 } // end function getCourseName
27
28 // display a welcome message to the GradeBook user
29 void GradeBook::displayMessage()
30 {
31    // call getCourseName to get the courseName
32    cout << "Welcome to the grade book for\n" << getCourseName()
33       << "!" << endl;
34 } // end function displayMessage
```

fig03_12.cpp

(2 of 2)

Outline

fig03_13.cpp

(1 of 1)

```
1  // Fig. 3.13: fig03_13.cpp
2  // GradeBook class demonstration after separating
3  // its interface from its implementation.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // function main begins program execution
11 int main()
12 {
13    // create two GradeBook objects
14    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
15    GradeBook gradeBook2( "CS102 Data Structures in C++" );
16
17    // display initial value of courseName for each GradeBook
18    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
19       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
20       << endl;
21    return 0; // indicate successful termination
22 } // end main
```

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

3.9 Separating Interface from Implementation (Cont.)

- **The Compilation and Linking Process**
  - **Source-code file is compiled to create the class's object code (source-code file must `#include` header file)**
    - **Class implementation programmer only needs to provide header file and object code to client**
  - **Client must `#include` header file in their own code**
    - **So compiler can ensure that the `main` function creates and manipulates objects of the class correctly**
  - **To create executable application**
    - **Object code for client code must be linked with the object code for the class and the object code for any C++ Standard Library object code used in the application**

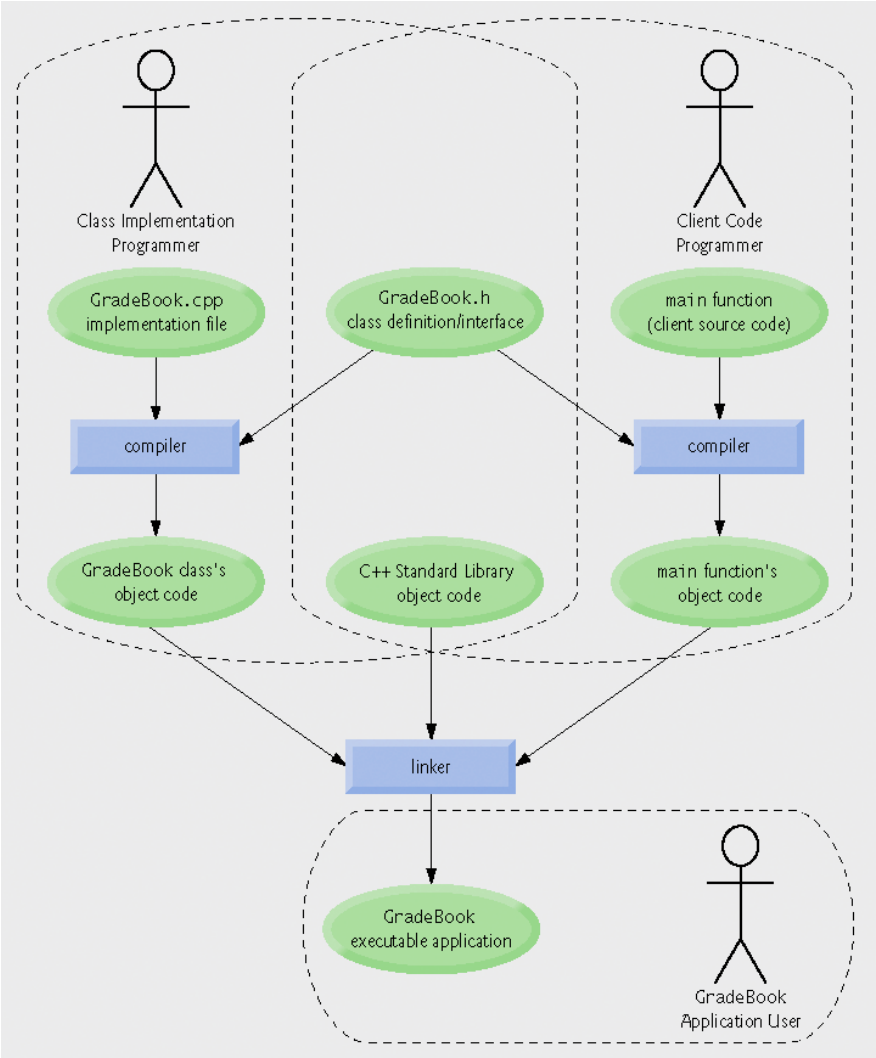

Fig.3.14 | Compilation and linking process that produces an executable application.

3.10 Validating Data with *set* Functions

- ***set* functions can validate data**
  - **Known as validity checking**
  - **Keeps object in a consistent state**
    - **The data member contains a valid value**
  - **Can return values indicating that attempts were made to assign invalid data**
- **`string` member functions**
  - **`length` returns the number of characters in the `string`**
  - **`Substr` returns specified substring within the `string`**

Outline

fig03_15.cpp

(1 of 1)

```
1  // Fig. 3.15: GradeBook.h
2  // GradeBook class definition presents the public interface of
3  // the class. Member-function definitions appear in GradeBook.cpp.
4  #include <string> // program uses C++ standard string class
5  using std::string;
6
7  // GradeBook class definition
8  class GradeBook
9  {
10 public:
11    GradeBook( string ); // constructor that initializes a GradeBook object
12    void setCourseName( string ); // function that sets the course name
13    string getCourseName(); // function that gets the course name
14    void displayMessage(); // function that displays a welcome message
15 private:
16    string courseName; // course name for this GradeBook
17 }; // end class GradeBook
```

Outline

fig03_16.cpp

(1 of 2)

```
1  // Fig. 3.16: GradeBook.cpp
2  // Implementations of the GradeBook member-function definitions.
3  // The setCourseName function performs validation.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13    setCourseName( name ); // validate and store courseName
14 } // end GradeBook constructor
15
16 // function that sets the course name;
17 // ensures that the course name has at most 25 characters
18 void GradeBook::setCourseName( string name )
19 {
20    if ( name.length() <= 25 ) // if name has 25 or fewer characters
21       courseName = name; // store the course name in the object
22
```

Constructor calls *set* function to perform validity checking

set functions perform validity checking to keep **`courseName`** in a consistent state

Outline

fig03_16.cpp

(2 of 2)

```
23    if ( name.length() > 25 ) // if name has more than 25 characters
24    {
25       // set courseName to first 25 characters of parameter name
26       courseName = name.substr( 0, 25 ); // start at 0, length of 25
27
28       cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
29          << "Limiting courseName to first 25 characters.\n" << endl;
30    } // end if
31 } // end function setCourseName
32
33 // function to get the course name
34 string GradeBook::getCourseName()
35 {
36    return courseName; // return object's courseName
37 } // end function getCourseName
38
39 // display a welcome message to the GradeBook user
40 void GradeBook::displayMessage()
41 {
42    // call getCourseName to get the courseName
43    cout << "Welcome to the grade book for\n" << getCourseName()
44       << "!" << endl;
45 } // end function displayMessage
```

Outline

fig03_17.cpp

```
1  // Fig. 3.17: fig03_17.cpp
2  // Create and manipulate a GradeBook object; illustrate validation.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "GradeBook.h" // include definition of class Gr
8
9  // function main begins program execution
10 int main()
11 {
12    // create two GradeBook objects;
13    // initial course name of gradeBook1 is too long
14    GradeBook gradeBook1( "CS101 Introduction to Programming in C++" );
15    GradeBook gradeBook2( "CS102 C++ Data Structures" );
16
```

Constructor will call *set* function to perform validity checking

Outline

fig03_17.cpp

(2 of 2)

```
17    // display each GradeBook's courseName
18    cout << "gradeBook1's initial course name is: "
19       << gradeBook1.getCourseName()
20       << "\ngradeBook2's initial course name is: "
21       << gradeBook2.getCourseName() << endl;
22
23    // modify myGradeBook's courseName (with a valid-length string)
24    gradeBook1.setCourseName( "CS101 C++ Programming" );
25
26    // display each GradeBook's courseName
27    cout << "\ngradeBook1's course name is: "
28       << gradeBook1.getCourseName()
29       << "\ngradeBook2's course name is: "
30       << gradeBook2.getCourseName() << endl;
31    return 0; // indicate successful termination
32 } // end main
```

Call *set* function to perform validity checking

```
Name "CS101 Introduction to Programming in C++" exceeds maximum length (25).
Limiting courseName to first 25 characters.

gradeBook1's initial course name is: CS101 Introduction to Pro
gradeBook2's initial course name is: CS102 C++ Data Structures

gradeBook1's course name is: CS101 C++ Programming
gradeBook2's course name is: CS102 C++ Data Structures
```

Software Engineering Observation 3.6

Making data members `private` and controlling access, especially write access, to those data members through `public` member functions helps ensure data integrity.

Error-Prevention Tip 3.5

The benefits of data integrity are not automatic simply because data members are made `private`—the programmer must provide appropriate validity checking and report the errors.

Software Engineering Observation 3.7

Member functions that *set* the values of `private` data should verify that the intended new values are proper; if they are not, the *set* functions should place the `private` data members into an appropriate state.

3.11 (Optional) Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document

- **Identifying the classes in a system**
  - **Key nouns and noun phrases in requirements document**
    - **Some are attributes of other classes**
    - **Some do not correspond to parts of the system**
    - **Some are classes**
      - **To be represented by UML class diagrams**

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.3.18 | Nouns and noun phrases in the requirements document.

3.11 (Optional) Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

- **Modeling classes with UML class diagrams**
  - **Top compartment contains name of the class**
  - **Middle compartment contains attributes**
  - **Bottom compartment contains operations**
  - **An elided diagram**
    - **Suppress some class attributes and operations for readability**
  - **An association**
    - **Represented by a solid line that connects two classes**
    - **Association can be named**
    - **Numbers near end of each line are multiplicity values**
    - **Role name identifies the role an object plays in an association**

Fig.3.19 | Representing a class in the UML using a class diagram.

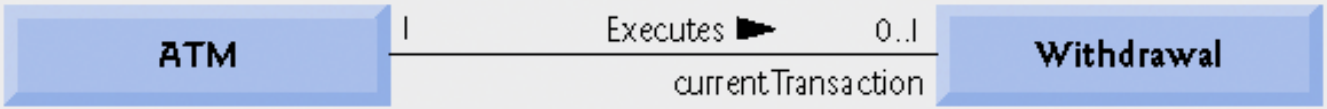

Fig.3.20 | Class diagram showing an association among classes.

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.3.21 | **Multiplicity types.**

3.11 (Optional) Software Engineering Case Study: Identifying the Classes in the ATM Requirements Document (Cont.)

- **Composition relationship**
  - **Indicated by solid diamonds attached to association lines**
  - **Composition properties**
    - **Only one class can represent the whole**
    - **Parts only exist while whole exists, whole creates and destroys parts**
    - **A part may only belong to one whole at a time**
- **Hollow diamonds indicate aggregation**
  - **A weaker form of composition**
- **Types of associations**
  - **One-to-one**
  - **One-to-many**
  - **Many-to-one**

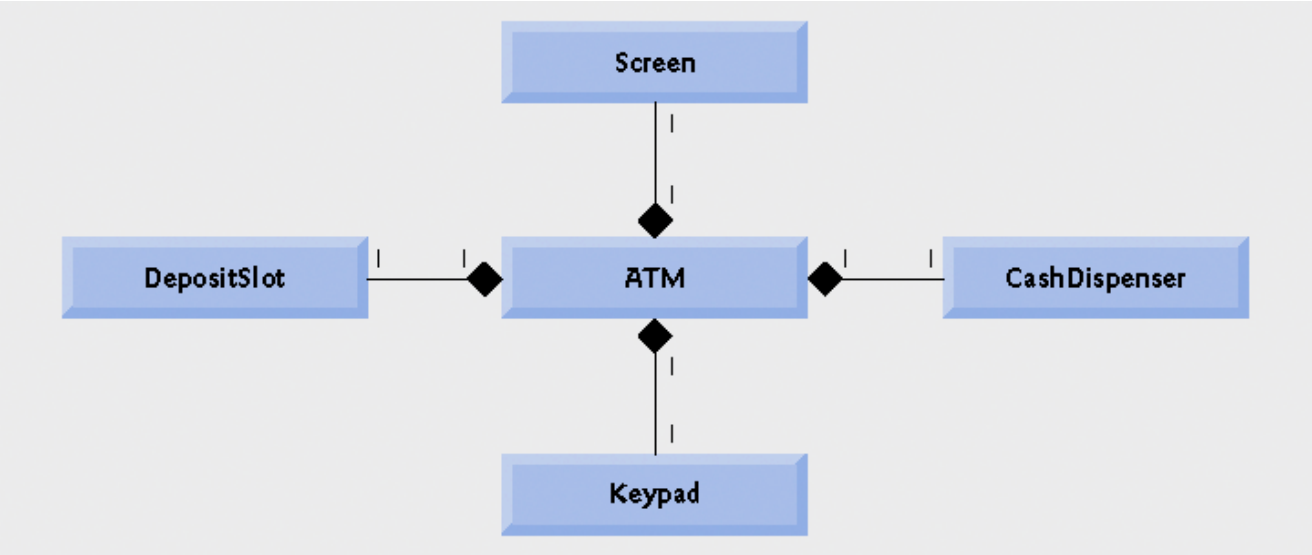

Fig.3.22 | Class diagram showing composition relationships.

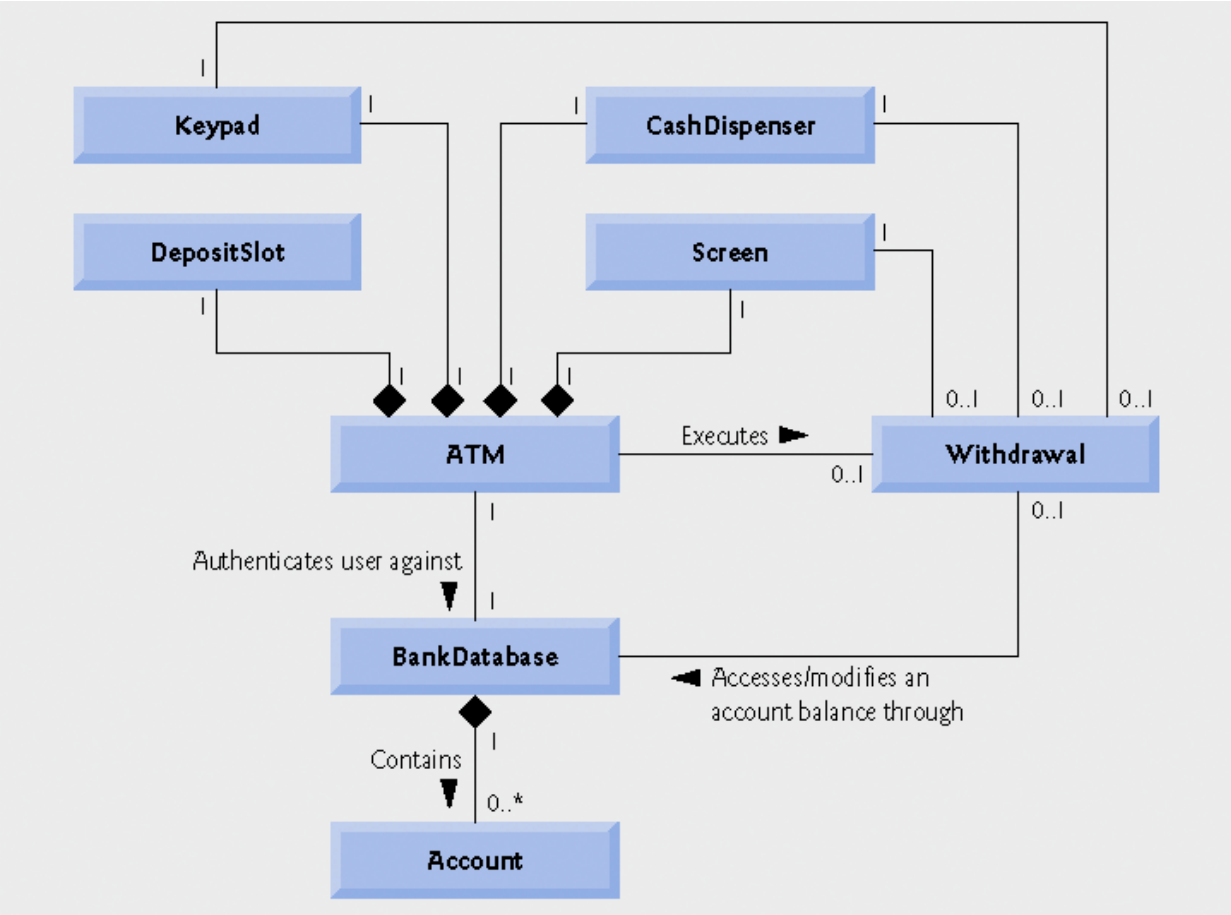

Fig.3.23 | Class diagram for the ATM system model

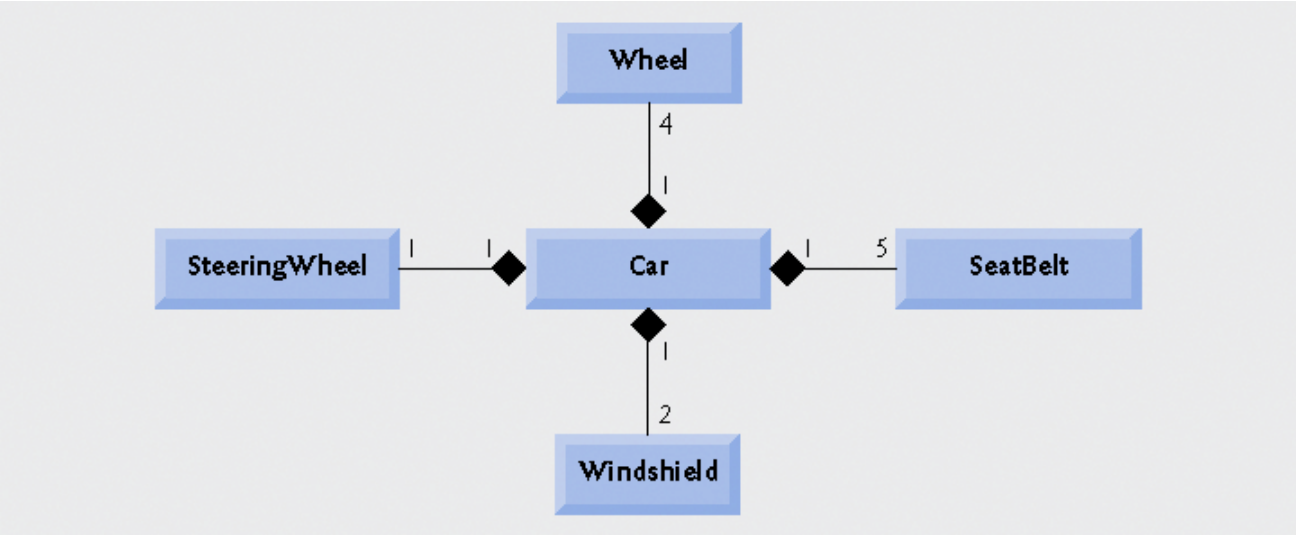

Fig.3.24 | Class diagram showing composition relationships of a class `Car`.

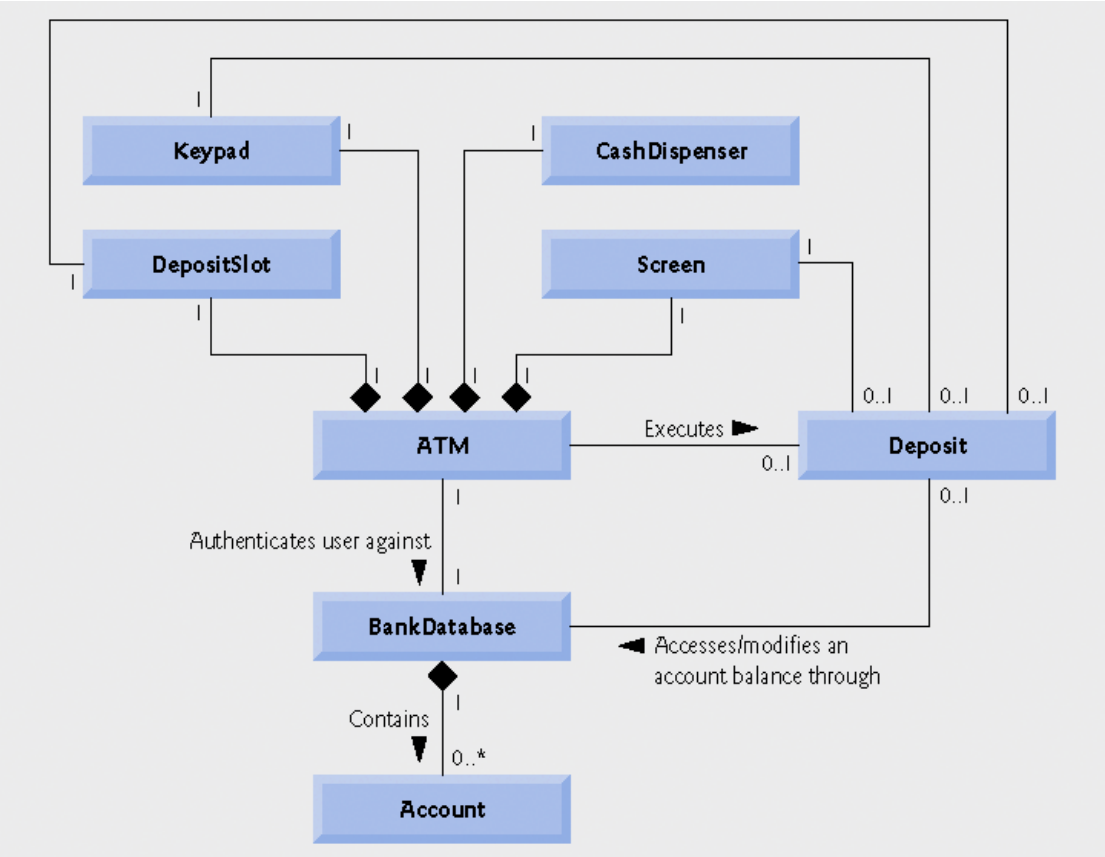

Fig.3.25 | Class diagram for the ATM system model including class `Deposit`.