5.3 Identification of Relationships

If we are constructing a class room of a college, it will consist of walls, windows, doors, fans, tube lights, projector, black board, etc. All of these things are related to each other. Walls are connected to each other. Doors are fixed inside the walls to provide way to people to enter the room. Windows are fixed for light or air purpose. Fans and tube lights are fixed on the ceilings of the wall to provide cooling and reduce darkness. Thus, all these things together form a class room. These things have different kinds of relationships amongst them. Similarly in OOA, the classes can connect to each other having different kinds of relationships. These relationships provide collaboration amongst classes.

A system consists of classes and objects. The system model depicts the communication amongst classes. The instances of a class communicate through sending messages. For example, the book is issued after the issue book message is received by the issue book object. Thus, a message is sent from one object to another. A relationship provides a channel through which instances of classes communicate with each other. They represent connection amongst instances of classes. There are five types of relationships amongst objects: association, aggregation, composition, dependency and generalization. Relationships along with their notations are summarized in Table 5.4.

Table 5.4 Relationships among objects

Relationship	Description	Notation
Association	It provides structural connections between instances of classes.	————
Aggregation	It provides whole-part kind of relationship between classes.	◇————
Composition	It provides strong aggregation between classes.	◆————
Dependency	It signifies that changes in one class affect the other class.	- - - - - - ->
Generalization	It signifies parent-child relationship amongst classes.	————▷

5.3.1 Association

Association is a structural connection between classes. This type of relationship is mostly bidirectional. In another term, the association relationship provides a link between different objects of the classes in the relationship. For example, there is a connection between IssueBookController class and Book class. This implies that the objects of IssueBookController class are linked with the objects of Book class. Binary associations are used to connect exactly two classes. The association relationship is represented by a solid line between the participating classes as shown in Figure 5.7.

An association relationship may be associated with a name. The name describes the type of relationship between two classes. For example, IssueBookController class manages Transaction class, as shown in Figure 5.8.

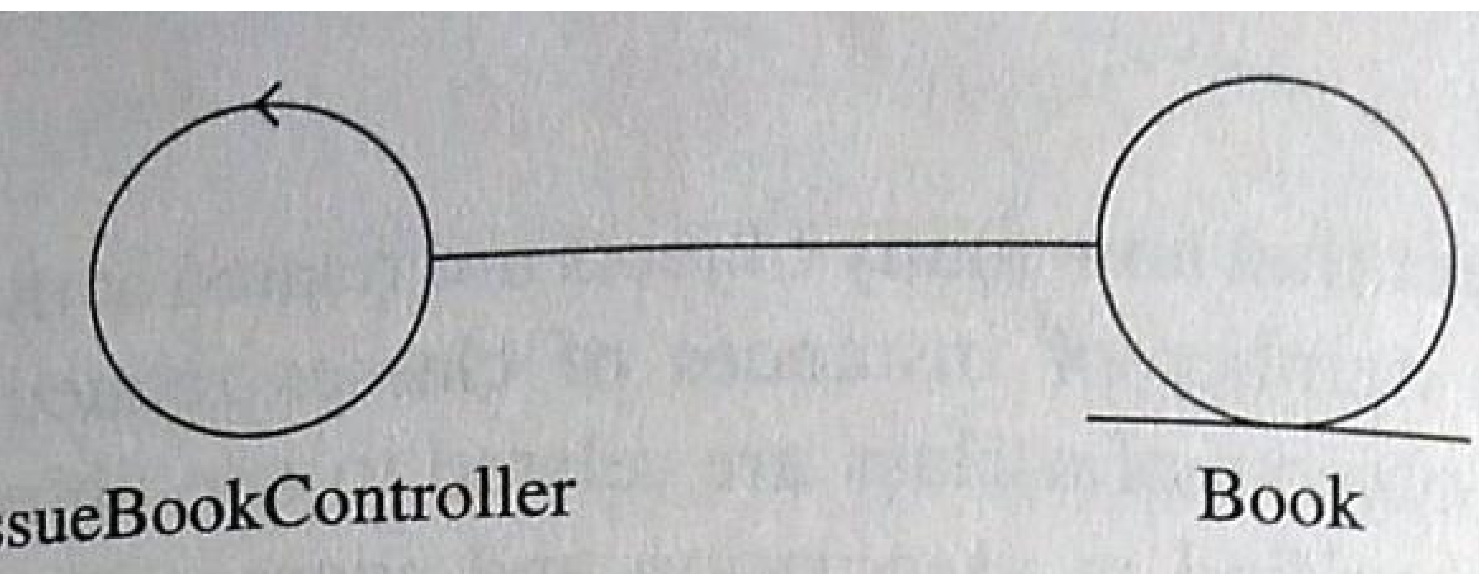

Figure 5.7 Association relationship.

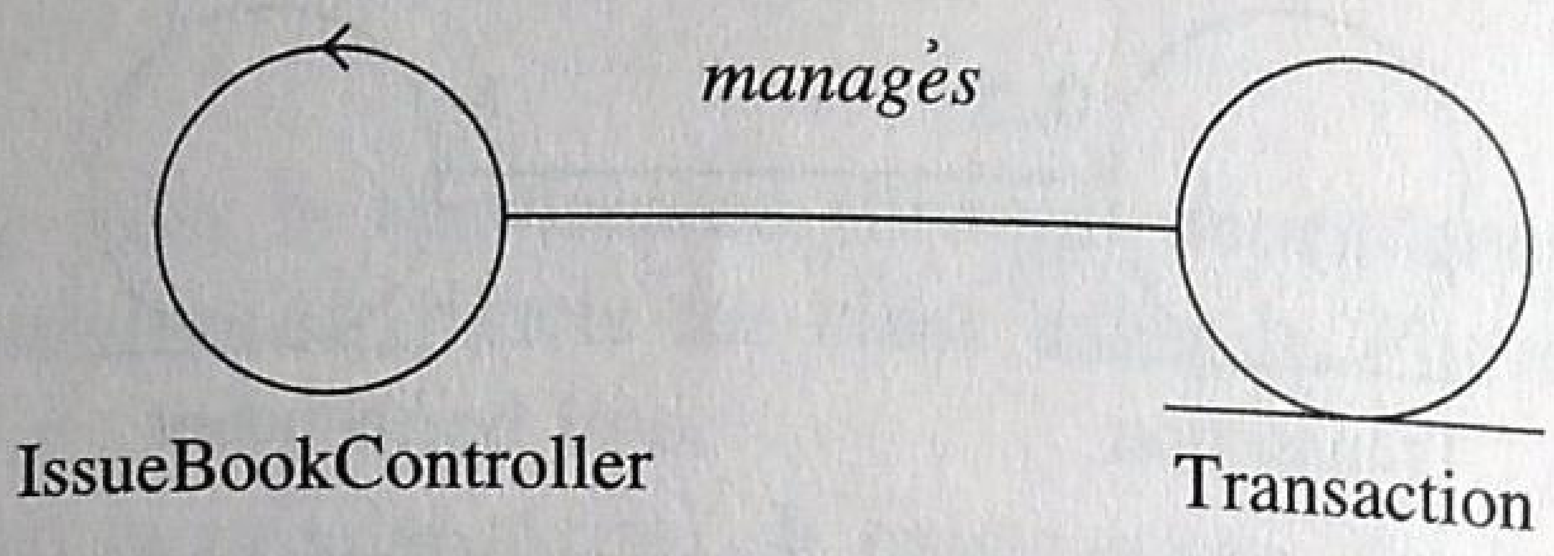

Figure 5.8 Association names.

When two classes are in association relationship, a role may also be associated between them. A role describes the function a class plays in an association. For example, a member may play the role of a student, faculty or employee when it is associated with Transaction class (see Figure 5.9). On the end of each association shown in Figure 5.9, multiplicity (refer to Section 5.3.3) is specified.

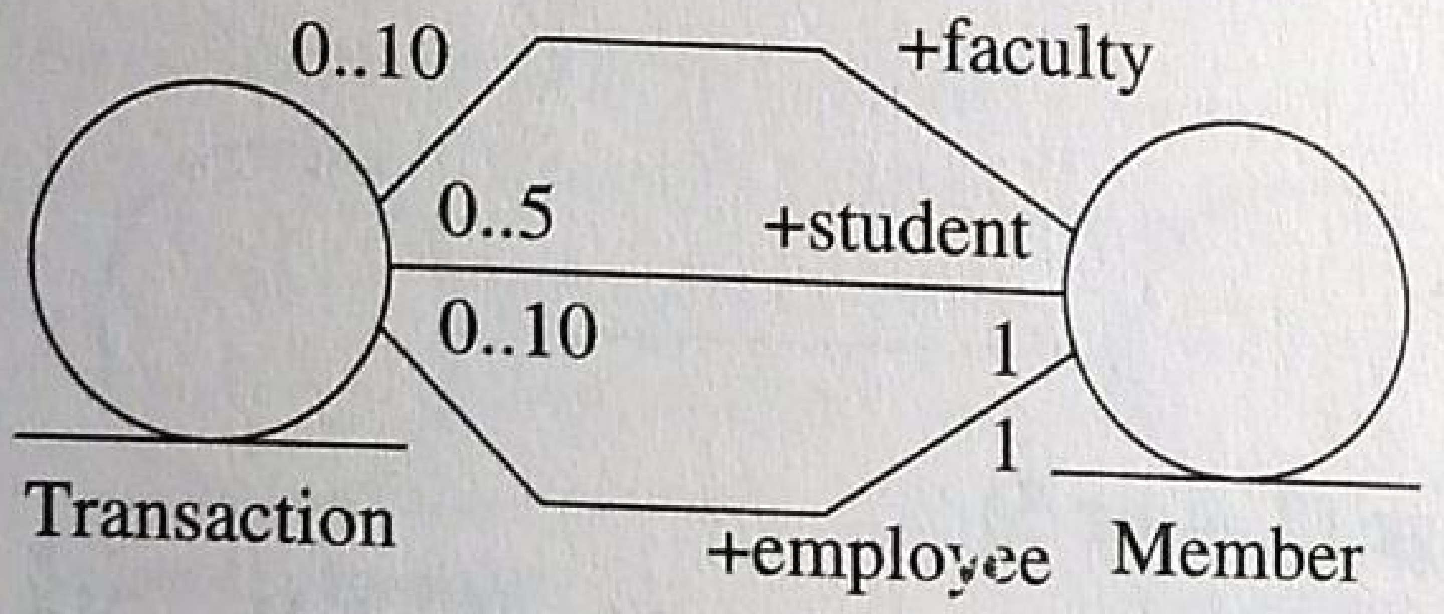

Figure 5.9 Association roles.

5.3.2 Aggregation

The aggregation relationship models the whole-part type of relationships. It depicts that a class is a part of another class. Some operations in the whole class may be applied to the other class. Aggregation represents 'has-a' relationship. The notation used for aggregation relationship is a line with a diamond at the end, i.e. to the class denoting the whole. For example, in the LMS, a book may be issued to a student/faculty/employee. The relationship between the Book class and the Library class is aggregation which means that the Book class is a part of the Library class as shown in Figure 5.10.

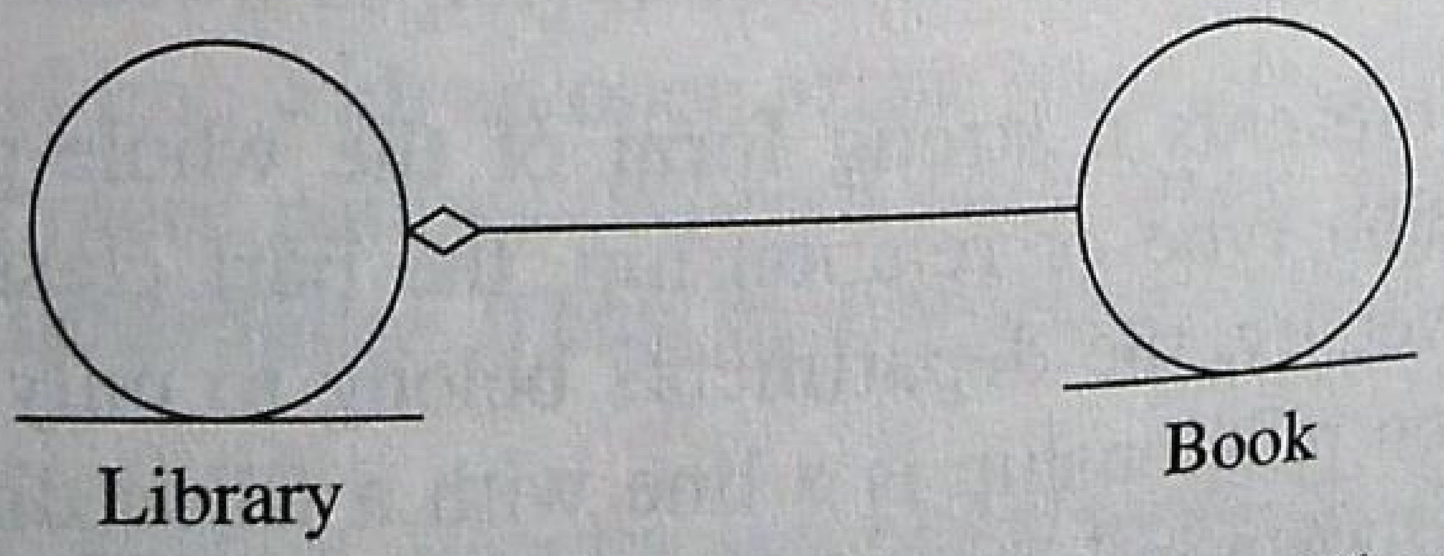

Figure 5.10 Aggregation relationship between library and book.

5.3.3 Multiplicity

We must specify for an object that how many objects are related at the other end of an association. Multiplicity represents the number of instances of classes related to one instance of another class. It depicts how many objects of a class are related to one object of another class at a given time. Multiplicity can be specified in association and aggregation relationships. Multiplicity is expressed by a range of values as shown in Figure 5.11.

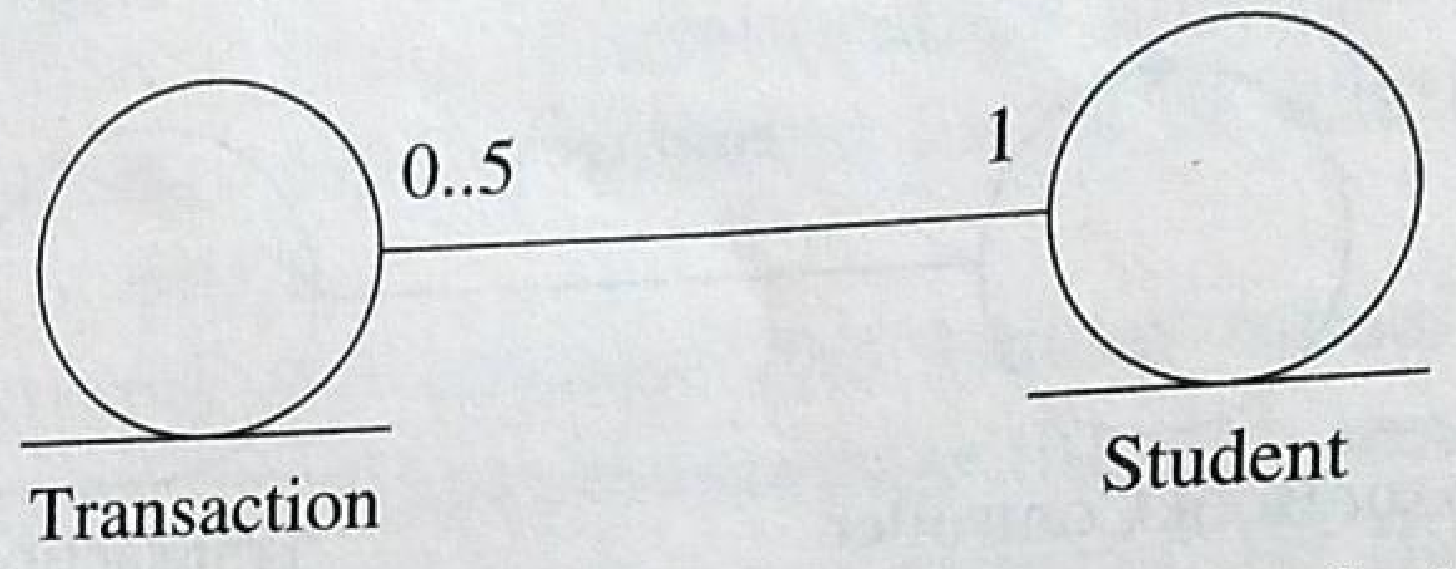

Figure 5.11 Multiplicity indicator of transaction and student association.

When the multiplicity is stated at one end of an association, it specifies the number of objects that must be there for each object of the class at the opposite end. Multiplicity can be shown as many (0..*), one or more (1..*) and exactly one (1). Hence, multiplicity is denoted as:

Minimum_value..maximum_value

Some commonly used multiplicity indicators are given as follows:

1	one
0..*	many
1..*	one or more
0..1	zero or one
3..5	specific range (3, 4 or 5)

The relationship shown in Figure 5.11 can be read in the following ways:

1. One transaction object is related to exactly one student. For example, B101 book is related to student Ram (a student object).
2. One student object may be related to zero to five transaction objects. For example, Ram (a student object) is related to B101, B105 and B111 (the student has three books issued). Since the range of multiplicity is zero to five, a minimum of zero and maximum of five books may be issued to a student.

Control classes usually depict one-to-one multiplicity (Boggs and Boggs, 2002). As the application is executed, only one IssueBookController object may be required.

5.3.4 Composition

Composition relationship represents a strong form of the whole-part relationship. When two classes are having composition type of relationship, the part class belongs to only the whole class. For example in a university, the departments belong to only the specific university. The notation used for composition relationship is a line with a filled diamond at the end, i.e. to the class denoting the whole as shown in Figure 5.12.

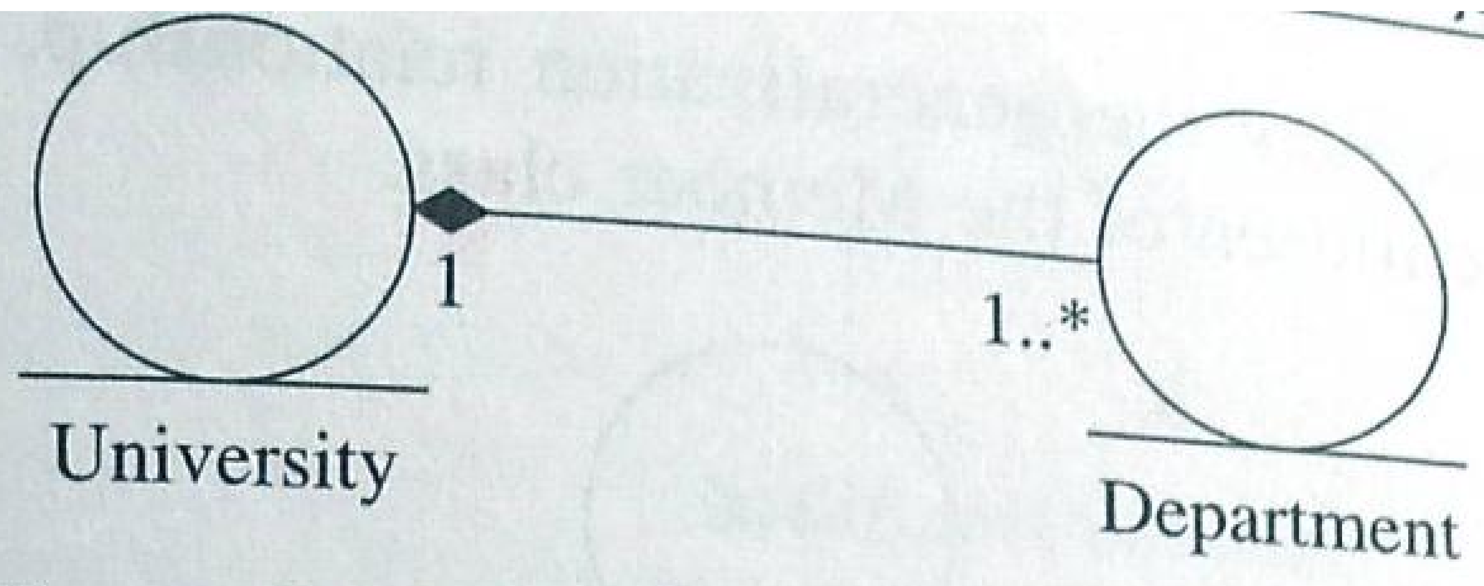

Figure 5.12 Composition relationship.

5.3.5 Dependency

The class referring another class is represented through dependency relationship. Hence, the change in the class being referenced affects the class using it. The dependency relationship is graphically depicted by a directed dashed line.

Dependency relationship is unidirectional. It exists when the instance of a class is either passed as a parameter variable in a method or used as a local variable in a method of another class.

When a system is designed, the aim is to minimize dependency relationships amongst classes. It increases complexity and decreases maintainability of the system. For example, a university maintains the record of programmes running in a school. We may consider a SchoolSchedule class which is dependent on programmes of the university as it uses the object of programme class as parameter to its member 'add'. Thus, the SchoolSchedule class is dependent on programme class as shown in Figure 5.13.

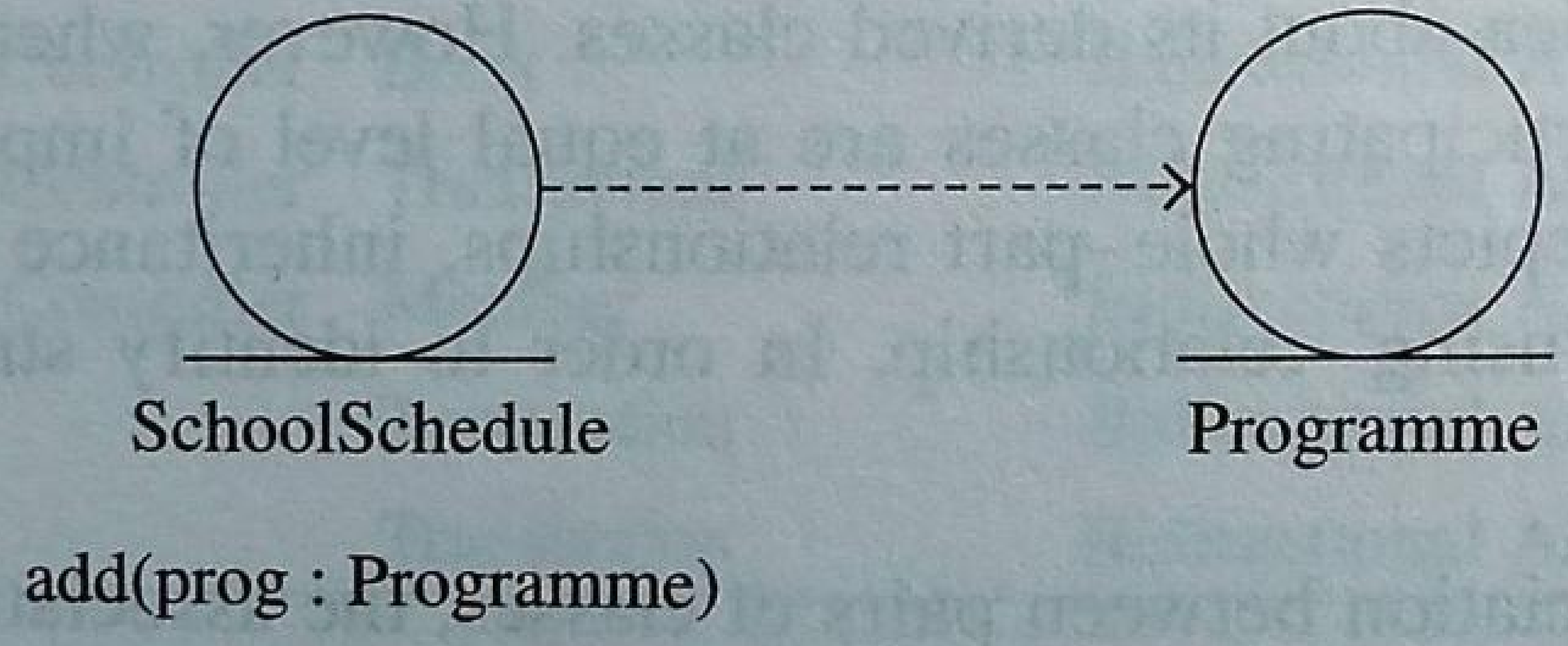

Figure 5.13 Dependency relationship.

5.3.6 Generalization

Generalization is a relationship between the parent class and the child class. This relationship depicts inheritance relationships. Generalization relation is also known as 'is-a' relationship. In this relationship, the child inherits all the properties of its parents but vice versa is not true. The child also consists of its own attributes and operations. When the operation of a child has the same name and signature as of its parents, it is known to override the operation of the parent class. This concept is known as *polymorphism*. The graphical notation of generalization relationship is shown as a solid line with an open arrow head that points towards the parent class.

A class can have zero or more parents. When a child inherits features from more than one class, it is known as multiple inheritance. Thus, in multiple inheritance a class has more than one parent class.

For example, in the LMS, Student, Faculty and Employee are all members of a library. They may have their own attributes but have some attributes in common which are placed in the base class Member. Figure 5.14 depicts the generalization relationship. Here, Student, Faculty and Employee entity classes are children of the Member class.

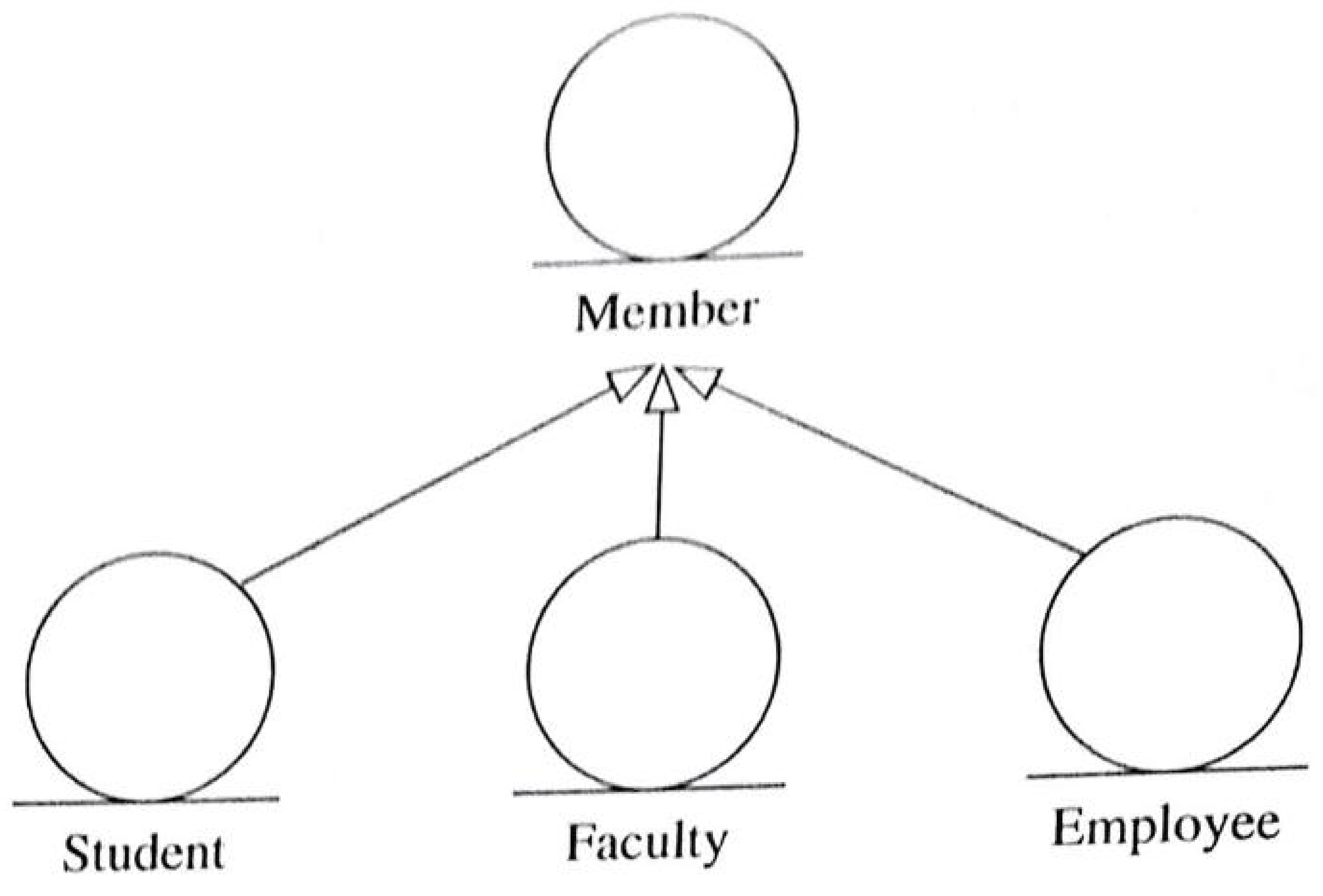

Figure 5.14 Inheritance relationship.

5.3.7 Modelling Relationships

When generalization and dependency relationships are modelled, the classes are not at equal level of importance. In the dependency relationship, one class is using the other class, but the other class has no knowledge of it. Similarly in the case of inheritance relationship, the base class does not have any idea about its derived classes. However, when association relationships are modelled, the two participating classes are at equal level of importance. Aggregation is a type of association that depicts whole–part relationships, inheritance depicts 'is-a' relationship and dependency depicts 'using' relationship. In order to identify structural relationships, the following may be seen:

- If there is an association between pairs of classes, the association relationship is used.
- If one object of a class passes messages to the other object other than through parameters of the method, then the association relationship is used.
- If a class is a whole and other classes are its parts, then an aggregation relationship is modelled. In other words, when a class is made up of other classes, the aggregation type of relationship is used.

The structural relationships can be identified by the use case descriptions explained in Chapter 3. Figure 5.15 models association relationships between classes involved in 'issue book' use case of the LMS and Table 5.5 shows their summary. The portion from the use case description of issue book use case is given as follows:

Basic Flow

1. Student/faculty/employee membership number is read through the bar code reader.
2. The system displays information about the student/faculty/employee.
3. Book information is read through the bar code reader.
4. The book is issued for the specified number of days and the return date of the book is calculated.
5. The book and student/faculty/employee information is saved into the database.

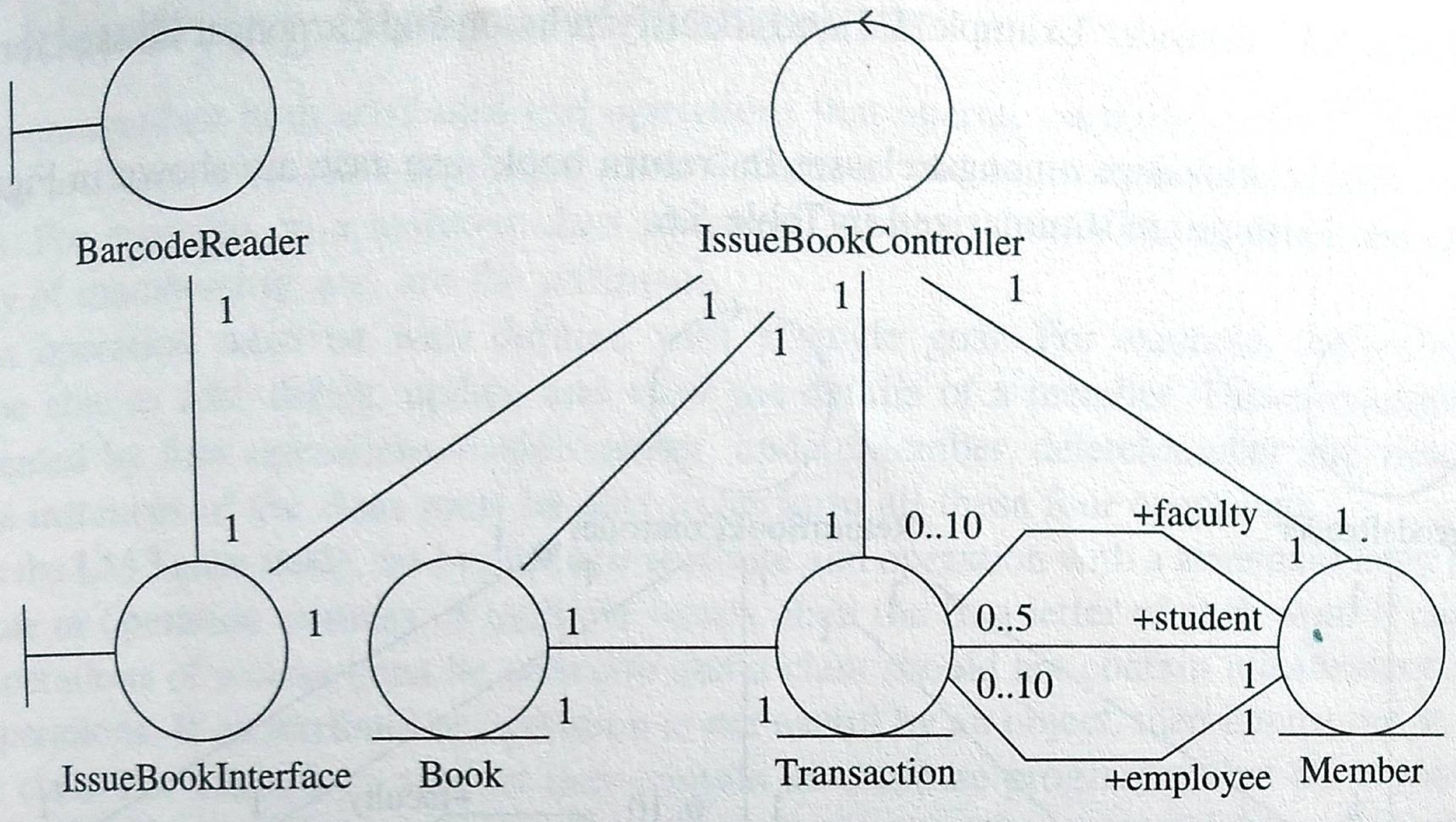

Figure 5.15 Relationships between classes in 'issue book' use case.

Table 5.5 Summary of relationships shown in Figure 5.14

Sending class	Receiving class	Relationship
BarcodeReader	IssueBookInterface	Bidirectional Association
IssueBookInterface	IssueBookController	Bidirectional Association
IssueBookController	Book	Bidirectional Association
IssueBookController	Transaction	Bidirectional Association
IssueBookController	Member	Bidirectional Association
Book	Transaction	Bidirectional Association
Member	Transaction	Bidirectional Association

Hence, all the relationships identified in 'issue book' use case are bidirectional association.

EXAMPLE 5.4 Consider Example 5.2 and identify relationships amongst classes for 'retur book' use case.

Solution The relationships amongst classes in 'return book' use case are shown in Figure 5.1 and these relationships are summarized in Table 5.6.

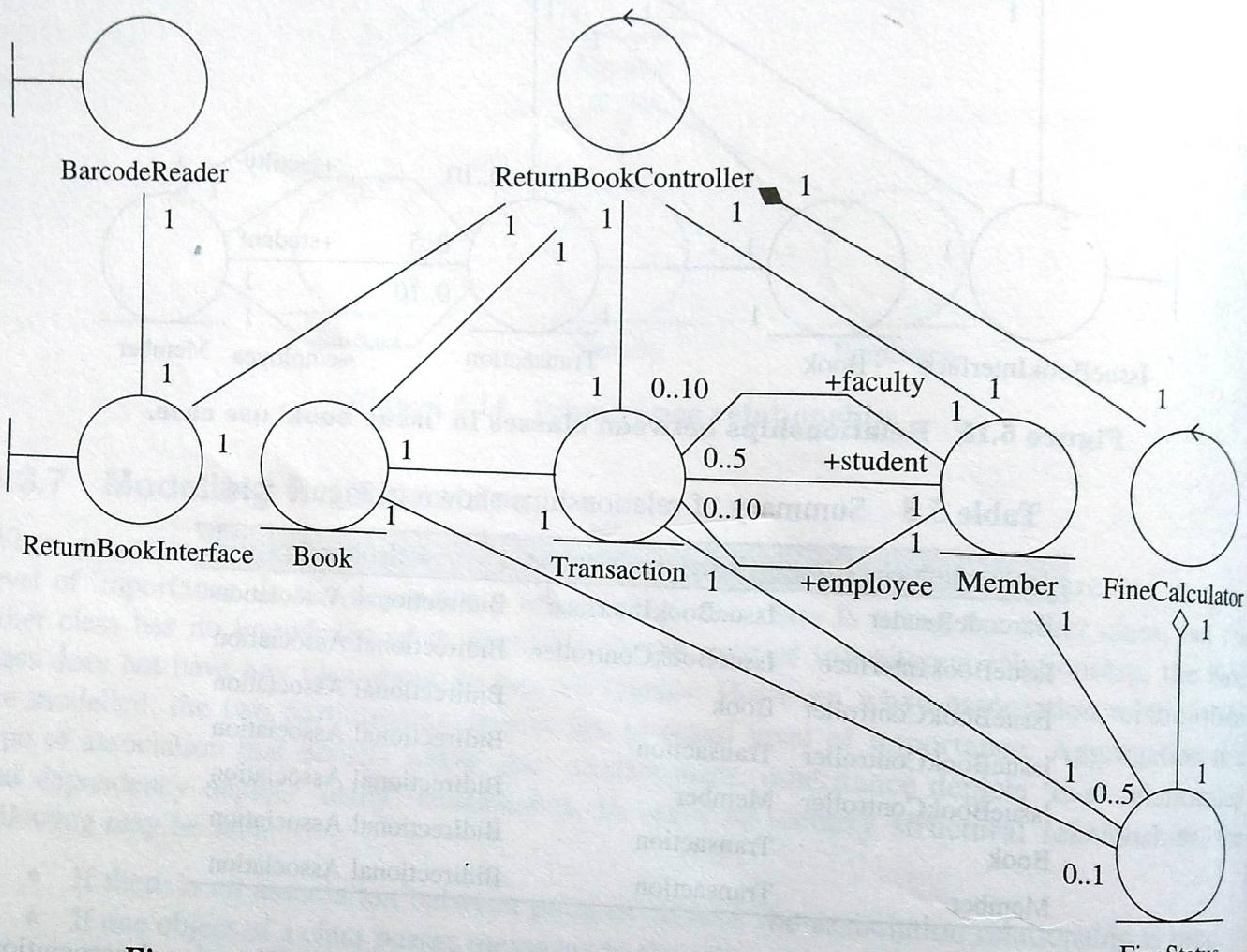

Figure 5.17 Relationships between classes in 'return book' use case.

Table 5.6 Summary of relationships shown in Figure 5.17

Sending class	Receiving class	Relationship
BarcodeReader	ReturnBookInterface	Bidirectional Association
ReturnBookInterface	ReturnBookController	Bidirectional Association
ReturnBookController	Book	Bidirectional Association
ReturnBookController	Transaction	Bidirectional Association
ReturnBookController	Member	Bidirectional Association
Book	Transaction	Bidirectional Association
Member	Transaction	Bidirectional Association
FineCalculator	ReturnBookController	Composition
FineCalculator	FineStatus	Aggregation

5.4 Identifying State and Behaviour

Classes encapsulate both attributes and operations that operate on those attributes, into a single unit. An attribute of a class represents data definition and is used to store information in the objects. For example, in a member class memberID, name, dateOfBirth, phone, email, date of validity of membership, etc. are the attributes.

An operation must be well defined with a single goal. For example, the member class must be able to add, delete, update and view the details of a member. These requirements are represented by four operations—addMember, updateMember, deleteMember and viewMember. All the instances of the class must be able to perform all these four operations.

In the LMS case study, we begin each attribute and operation with a lowercase letter and if the attribute or operation consists of multiple words, then the first letter of each word is capitalized. The operations of a class must be cohesive and a class should not contain unnecessary attributes and operations. If an attribute or operation is not useful by an object, then it must not be included in the class. For example, a student may contain an attribute programme but the student is only concerned with single programme. The programmes must be managed by a different school class. The procedure for identification of attributes and operations is described in subsequent sections.

5.4.1 Attributes

Attributes represent the information to be stored. The entity objects may contain many attributes. Attributes can be identified from the use case description. For example, add a book subflow of Maintain Book Details use case is given as follows:

Basic Flow 1: Add a book

The system requests that the administrator/DEO enter the book information. This includes:

- Book bar code ID
- Accession number
- Subject descriptor
- ISBN
- Title
- Language
- Author
- Publisher

Once the administrator/DEO provides the requested information, the book is added to the system.

This subflow shows that the Book entity class must contain the attributes such as book bar code ID, accession number, subject descriptor, ISBN, title, language, author, and publisher. This information needs to be stored in an entity class Book which is required to keep the details of the number of books in a library. Hence, the attributes can be discovered by extracting the elements that are needed from the use case description. They can also be identified by the class description and from the expertise of the analyst. In addition to the attributes identified by use case description of subflow 'add a book', one additional attribute that keeps the record of the issue status of the book must be kept. Scope and domain expertise are used to determine this type of attribute. This attribute may not be alive in the use case but is required to keep track of the issue status of the book. The UML representation of an entity class Book is shown in Figure 5.18 along with its attributes. Each attribute consists of a data type such as integer, real or string as shown in Figure 5.18. An attribute may also be a complex data type.

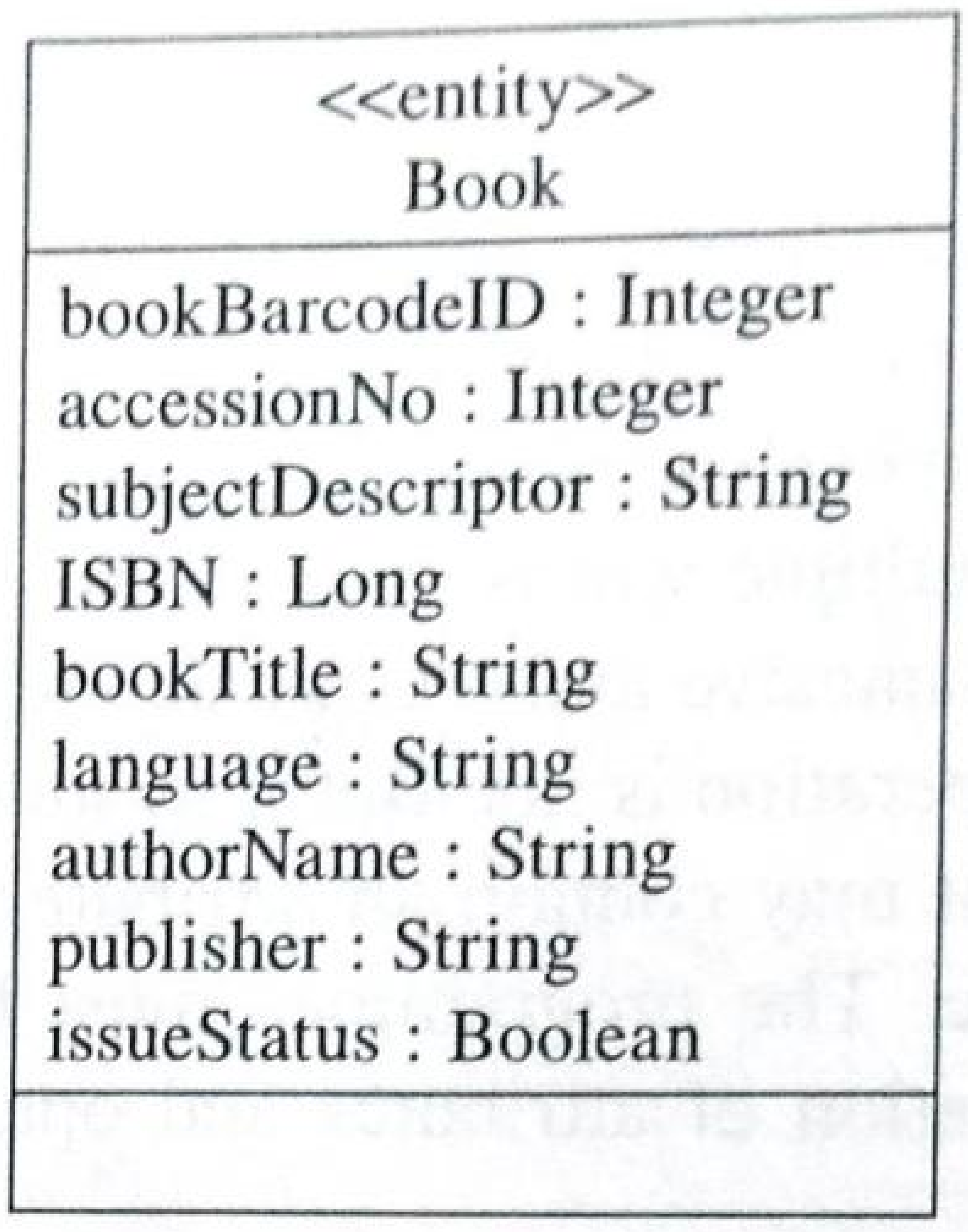

Figure 5.18 Attributes in a Book class.

The attributes in the Book class can also be identified from section 3.1.1 user interfaces of the SRS. The portion of the details of the book form from the SRS is given as follows:

Various fields available on this form will be:

- **Book bar code ID:** Numeric
- **Accession number:** Numeric and will have value from 10 to 99999.
- **Subject descriptor:** Will display all the subject descriptors.
- **ISBN:** Numeric
- **Title:** Alphanumeric of length 3 to 100 characters. Special characters (except brackets) are not allowed. Numeric data will be allowed.
- **Language:** Will display all the languages available.
- **Author:**
  - ♦ **First name:** Alphanumeric of length 3 to 50 characters. Special characters are not allowed. Numeric data will not be allowed.
  - ♦ **Last name:** Alphanumeric of length 3 to 50 characters. Special characters are not allowed. Numeric data will not be allowed.
- **Publisher:** Alphanumeric of length 3 to 300 characters. Special characters (except brackets) are not allowed. Numeric data will be allowed.

The above portion of the SRS shows the description of the fields of user interface 'book form'. These fields need to be stored, hence should be converted as attributes of Book class in the OOA phase.

Hence, there are four methods to identify the attributes in a class:

1. The information to be stored can be extracted from the use case description.
2. The fields in the user interfaces given in the SRS document can be used to extract the attributes.