# 5

# Object-Oriented Analysis

After requirements have been captured, the analysis of the identified requirements begins. The process of analysis involves construction of logical structure of a system that can be maintained during the software development life cycle. Object-oriented analysis (OOA) identifies and defines the real-world objects that are involved in interaction with the system. It also identifies the objects that can be used to provide alternative solutions to the problem. The real-world objects are defined in terms of classes, attributes and operations. These objects, thus, can be mapped into objects in object-oriented languages easily. The OOA constructs a robust and ideal model that is free from the details of implementation environment. Hence, the OOA is the process of defining the problems in terms of real-world objects. It gives a high level of understanding of possible solution to the requirements identified.

The aim of this chapter is to extract classes and attributes. The types of classes are identified and relationships amongst the identified classes are discovered. Finally, the operations and attributes in the classes are defined. All these concepts are explained along with a real-life case study of LMS.

## 5.1 Structured Analysis versus Object-Oriented Analysis

Structured analysis (SA) involves converting the requirements into specifications. Pressman (2005) defines structured analysis as:

> *Structured analysis (SA) takes a distinct input-process-output view of requirements. Data are considered separately from the processes that transform the data. System behavior, although important, tends to play a secondary role in structured analysis. The structured analysis approach makes heavy use of functional decomposition (partitioning of the data flow diagram).*

The SA involves top-down decomposition of the system into modules. It centres around the creation of process model using data flow diagram (DFD) and design of data model using





entity-relationship (ER) model. The DFD depicts the flow of data through the system. It shows the processes that need to be designed and finally developed. The focus of the DFD is on functions rather than on data. However, the data can be modelled using ER models. The disadvantages of function-oriented and data-oriented techniques are that the functions are given more importance than the data and the data and functions are modelled separately. Data and functions are collective activities, i.e. the data can be modified through functions.

In the OOA phase, the class modelling techniques are used which combine the data and functions into a single class. Thus, data and functions are given equal importance. The attributes in a class can only be accessed by the operations in the classes which prevent the accidental modification of data (attributes). The data are known as attributes and the functions are called operations. These are easily modifiable and maintainable, and can be directly refined in object-oriented design. In the OOA phase, relationships between classes are depicted through visual diagrams. The purpose of such diagrams is to:

- understand, extract and define the customer's requirements.
- serve as a basis for development of the software. Based on these diagrams, it would be easier to develop the database schema of the system and partially identify its operations.

## 5.2 Identification of Classes

A class is a collection of objects, with common attributes and operations. The objects with common attributes and operations have similar relationship to the class. Hence, a class is a template that groups attributes and operations together, and is used to create objects. Identification of classes is not a simple task. In the rational unified process, classes are categorized into three types: entity class, interface class and control class. These three types of classes allow the analyst to separate the functionality of the system and simplify the identification process. The OOA process is iterative. Hence, we keep on modifying the list and structure of classes, as the analysis and design progresses.

### 5.2.1 Entity Classes

Entity classes include those classes that are going to persist longer into the system. These are the classes which have to be stored and maintained for a longer period, sometimes for the lifetime of the system. Figure 5.1 shows the notation for representing entity classes.

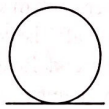



**Figure 5.1 Notation for an entity class.**

Entity classes contain information needed to accomplish a given task. These may be identified by extracting the information that is required to complete a task from the use case description. Entity classes are also called *domain classes*. Consider the 'issue book' use case of

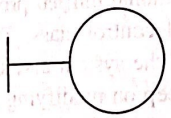
the LMS case study given in Chapter 3. The purpose of the use case is to allow the members (student/faculty/employee) to issue book from the university library. The use case requires the information of member, book and transaction. Hence, we identify three entity classes:

- Member
- Book
- Transaction

The instances of entity classes are known as entity objects. Entity classes may not be specific to one use case. For example, in the LMS, student class may be included in 'issue book' use case as well as in 'maintain student details' use case. The attributes and operations of an entity class may be identified by the actor of the use case. Entity classes represent the key attributes of the system that is to be developed. For instance, in an ATM system, customer and account are some entity classes and in a result management system, students, marks and courses are entity classes. The major source of information for extracting entity classes is the data fields of interface screens described in the SRS document.

### 5.2.2 Interface Classes

Interface classes handle the interaction in the system. They provide interface between the actor (human being or external system) and the system. These classes are known as *boundary classes*. These classes are used to model windows, buttons, communication protocols, etc. Usually, the life of the interface classes is only as long as the use case exists. Figure 5.2 shows the notation for representing interface classes.



Interface class

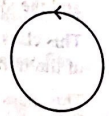
**Figure 5.2 Notation for an interface class.**

The interaction of actor with each use case scenario is examined to extract the interface classes. The interface classes represent the number of interfaces required by an actor to interact with all the paths in the use cases. For example, in the LMS, in order to issue a book, an interface is required. The actor library staff interacts with 'issue book' use case. This means the student/faculty/employee must be able to interact with something to issue book. A class containing all the options to enable the actor 'library staff' issue book will satisfy the given requirement. This class is called IssueBookInterface. BarcodeReader class will also be required to get the bar code details of the book and the library member. Unlike entity and control classes, interface classes are dependent on the surroundings of the system. If there is a change in the interface, then the interface class should change but the entity and control classes will remain unaffected.

### 5.2.3 Control Classes

Control classes are responsible for coordinating and managing entity and interface classes. The functionality that could not be placed in either of entity and interface classes may be placed in a control class. The control class is used to put together things so that a use case is completed. Control classes represent the dynamics of the system and handle the tasks and sequence of events. Figure 5.3 shows the notation for representing control classes.





Control class

**Figure 5.3 Notation for a control class.**

A control object (an instance of control class) is created when the use case starts and it ends when the use case finishes. Control classes may be identified from the use cases. One control class may be assigned to one use case. These classes are used to manage and coordinate the flow of events needed to accomplish the functionality of a given use case. If the control class is doing more than coordinating the sequence of events in the use case, then it must be reconsidered. For example, in the LMS, the data entry operator adds a student. The information about the procedure of addition of the student must be available with student class rather than StudentManager class. Now we will identify control classes for 'issue book' use case in the LMS. A control class can be added to handle the flow of events for 'issue book' use case. This class is called IssueBookController.

Some use cases in the system (specifically data manipulation use cases) may function without any control class. Examples of control classes include TransactionManager, LoginController, SecurityManager, ExceptionHandler. The control classes allow the entity classes to be more independent so that they can be used across multiple use cases. Control classes have the following properties:

1. They are independent of their surroundings.
2. They are used to control the flow of events.
3. The changes required in the control class are due to the changes in the entity classes.
4. They may not perform in the same manner each time they are executed.

Thus, the classes identified for 'issue book' use case in the LMS are summarized in Table 5.1 and their graphical notations are shown in Figure 5.4.

**Table 5.1** Summary of classes in 'issue book' use case

| Class name | Class type | Description |
|---|---|---|
| Member | Entity | This class is used to represent the information of the members (student/faculty/employee) in the university. |
| Book | Entity | This class is used to represent the information of the books in the university library. |
| Transaction | Entity | This class is used to represent the information of the transactions to the members in the library. |

(*Contd.*)

**Table 5.1** Summary of classes in 'issue book' use case (*Contd.*)

| Class name | Class type | Description |
|---|---|---|
| IssueBookInterface | Interface | This class provides interface between the actor and the system. |
| BarcodeReader | Interface | This class reads bar code of books and members of the library. |
| IssueBookController | Control | This class manages and controls the operations in the 'issue book' use case. |

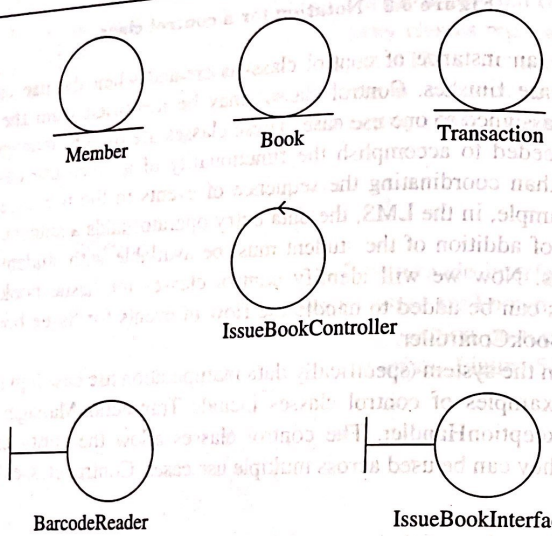



**Figure 5.4 Classes for 'issue book' use case.**

**EXAMPLE 5.1** Consider the case study of LMS, given in Chapter 3, Section 3.1. Identify entity classes, interface classes and control classes for 'login' use case.

***Solution*** In the login use case, one entity class 'login' containing the login details will be required. The summary of classes identified is given in Table 5.2 and these are diagrammatically depicted in Figure 5.5.

**Table 5.2** Summary of classes in 'login' use case

| Class name | Class type | Description |
|---|---|---|
| Login | Entity | This class is used to represent the login information of the operators of the system. |
| LoginInterface | Interface | This class provides login interface between the actor and the system. |
| LoginController | Control | This class manages and controls the operations in the 'login' use case. |



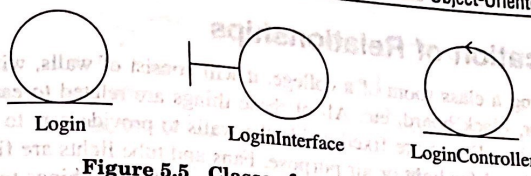



**Figure 5.5 Classes for 'login' use case.**

**EXAMPLE 5.2** Consider the case study of LMS, given in Chapter 3, Section 3.1. Identify entity classes, interface classes and control classes for 'return book' use case.

***Solution*** In the return book use case, three entity classes, two control classes and one interface class will be required. The summary of classes identified is given in Table 5.3 and they are diagrammatically depicted in Figure 5.6.

**Table 5.3** Summary of classes in 'return book' use case

| Class name | Class type | Description |
|---|---|---|
| BarcodeReader | Interface | This class reads bar code of books and members of the library. |
| Member | Entity | This class is used to represent the members of the library. |
| Book | Entity | This class is used to represent the information of the books in the university library. |
| Transaction | Entity | This class is used to store the information of the transactions to the members in the library. |
| ReturnBookInterface | Interface | This class provides return book interface to the actor of the system. |
| ReturnBookController | Control | This class manages and controls the operations in the 'return book' use case. |
| FineCalculator | Control | This class manages and controls the operations in the 'fine calculation' use case. |
| FineStatus | Entity | This class stores the fine amount and the status of the fine. |

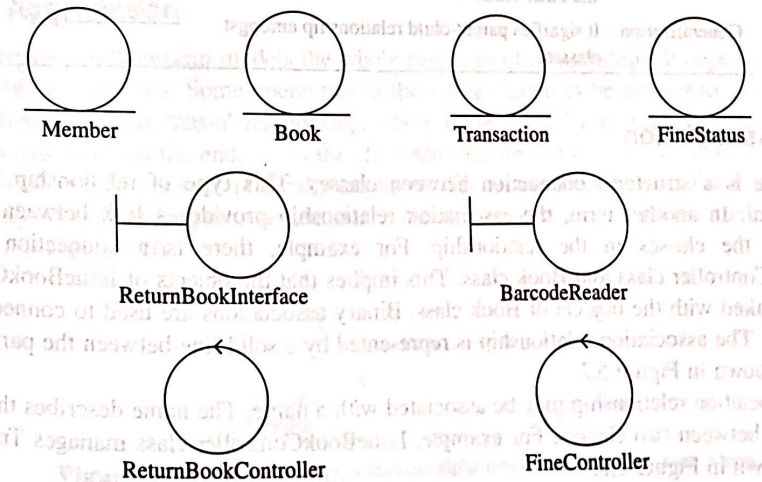



**Figure 5.6 Classes for 'return book' use case.**

## 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.



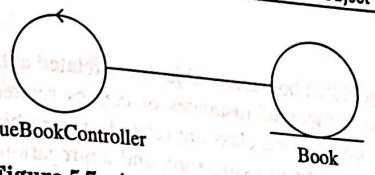



**Figure 5.7 Association relationship.**

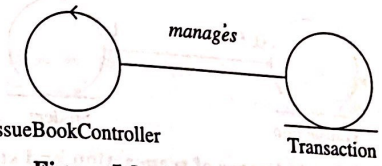



**Figure 5.8 Association names.**

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.

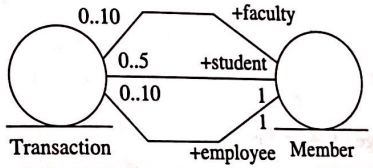



**Figure 5.9 Association roles.**

### 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.

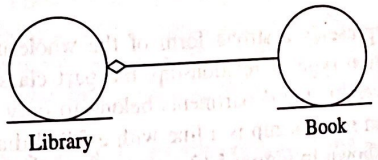



**Figure 5.10 Aggregation relationship between library and book.**

### 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.

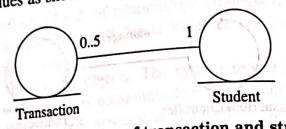



**Figure 5.11 Multiplicity indicator of transaction and student association.**

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.



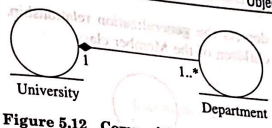



**Figure 5.12 Composition relationship.**

### 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.

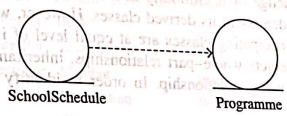



**Figure 5.13 Dependency relationship.**

### 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.

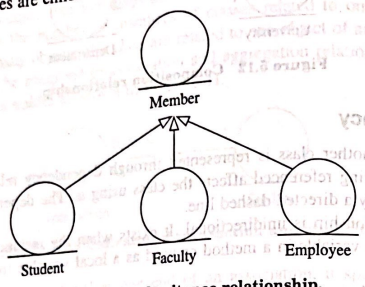



**Figure 5.14 Inheritance relationship.**

### 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.



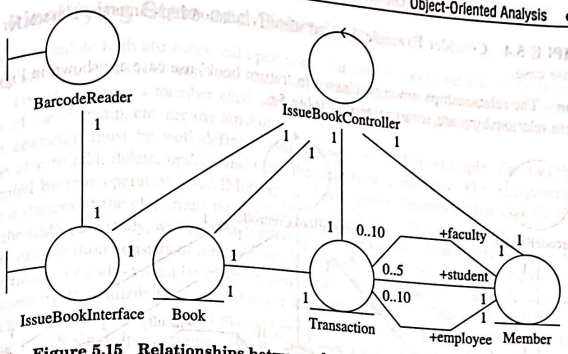



**Figure 5.15 Relationships between classes in 'issue book' use case.**

**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 5.3** Consider Example 5.1 and identify the relationships amongst classes for 'login' use case.

***Solution*** The relationships amongst various classes for 'login' use case are shown in Figure 5.16

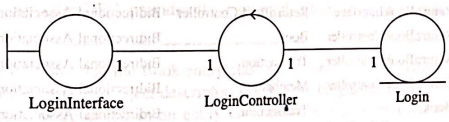



**Figure 5.16 Relationships between classes in 'login' use case.**

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

***Solution*** The relationships amongst classes in 'return book' use case are shown in Figure 5.17 and these relationships are summarized in Table 5.6.

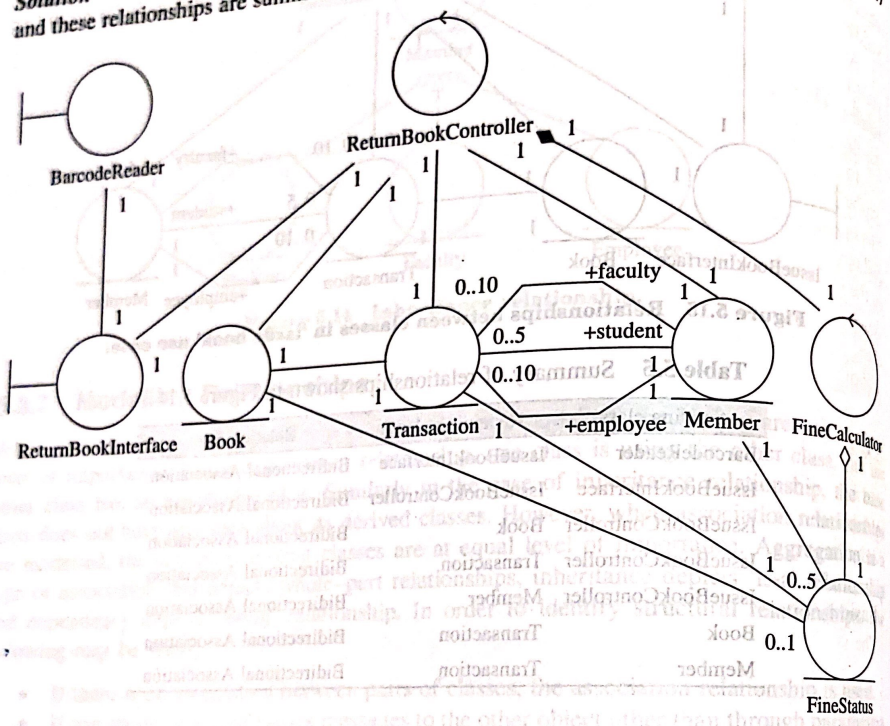



**Figure 5.17 Relationships between classes in 'return book' use case.**

**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 |