## 6.1 OVERVIEW OF OBJECT-ORIENTED CONCEPTS

Some important concepts and terms related to the object-oriente approach are depicted in Fig. 6.1. We first discuss the basic mechanism shown in the figure, viz. objects, classes, inheritance, messages, an methods. We then discuss some key concepts and look at some relate technical terms.

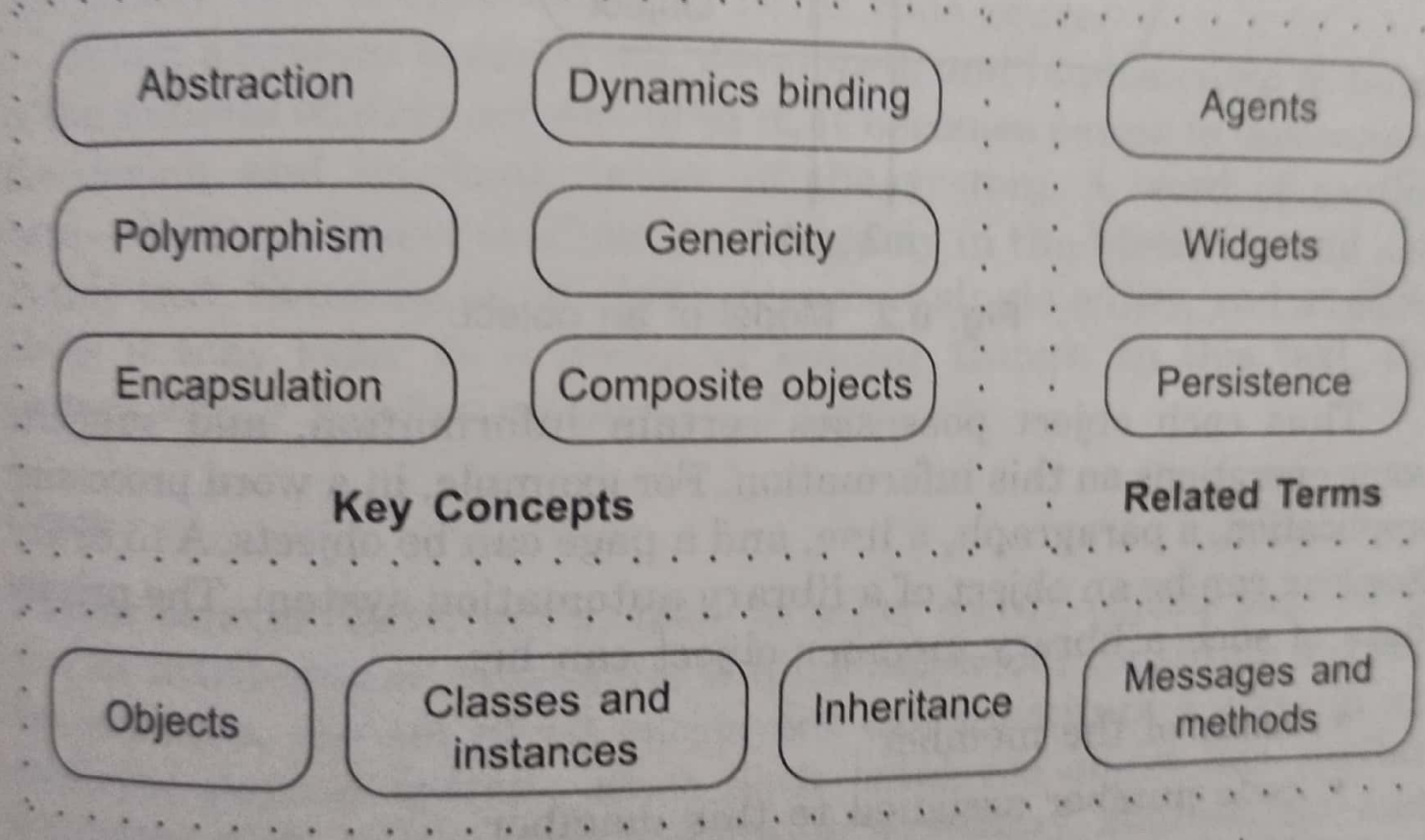


**Fig. 6.1** Important concepts in object-oriented approach.

### 6.1.1 Basic Mechanisms

#### Objects

In the object-oriented approach, a system is designed as a set of interacting objects. Usually, each object represents some tangible real-world entity such as a library member, an employee, a book, etc. However, sometimes we might consider certain conceptual entities as objects (e.g. a scheduler) to simplify solution to certain problems.

Each object essentially consists of some data that is private to

the object and a set of functions (or operations) that operate on those data (see Fig. 6.2). In fact, the functions of an object have the sole authority to operate on the private data of that object. In other words, an object cannot directly access the data internal to another object. However, an object can access the internal data of other objects by invoking the operations (i.e. methods) supported by those objects. In other words, each object can also be viewed as a data abstraction entity.

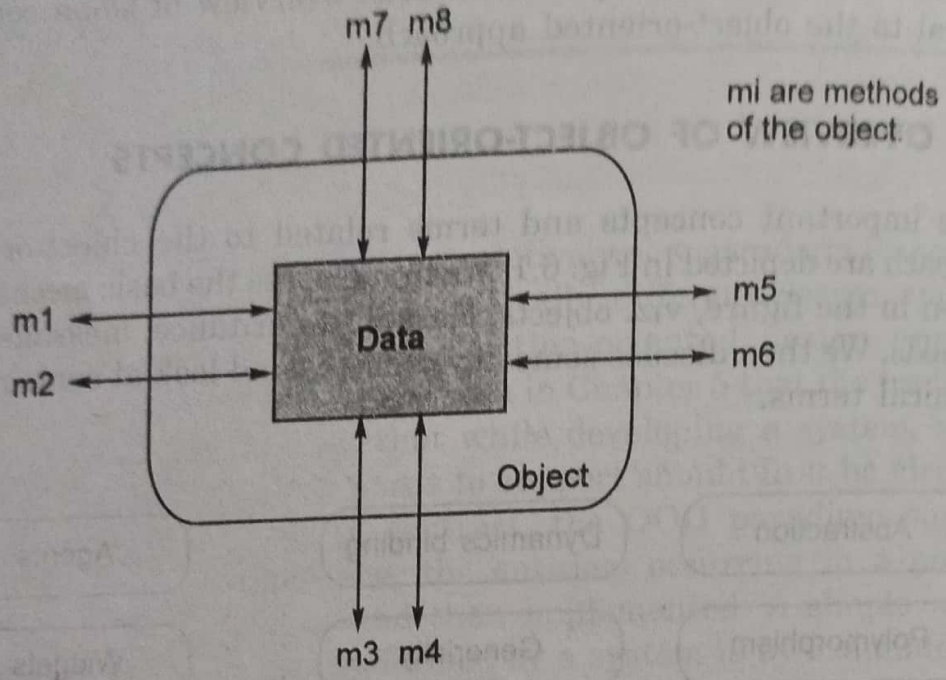


**Fig. 6.2** Model of an object.

Thus each object possesses certain information, and supports some operations on this information. For example, in a word processing application, a paragraph, a line, and a page can be objects. A `Library Member` can be an object of a library automation system. The private data of such a library member object can be:

- name of the member
- code number assigned to this member
- his date of birth
- his phone number
- his e-mail number
- date when he was admitted as a member
- his membership expiry date
- books issued to him, etc.

The operations supported by an employee object can be:

- issue-book
- find-books-due

### Class

Similar objects constitute a class. In other words, objects possessing similar attributes or displaying similar behaviour constitute a class. For example, the set of all employees can constitute a class in an employee payroll system, since each employee object has similar attributes such as his name, code number, salary, address, etc. and exhibits similar behaviour as other employee objects. Once we define a class it serves as a template for object creation. Thus classes can be considered as abstract data types (ADTs). In fact, each object must be defined as an *instance* of some class. In other words, the attributes and behaviour of an object are determined by the class it belongs to.

### Methods and Messages

We have already seen that the operations supported by an object are called its *methods*. Methods are the only means available to other objects for accessing and manipulating the data private to an object.

The methods of an object are invoked by sending messages to it. The set of valid messages to an object constitutes its *protocol*.

### Inheritance

The inheritance feature allows us to define a new class by extending or modifying an existing class. The original class is called the *base class* (or superclass) and the new class obtained through inheritance is called the *derived class* (or subclass). In Fig. 6.3, `Library Members` is the base class for the derived classes `Faculty`, `Students`, and `Staff`. Similarly, `Students` is the base class for the derived classes `Undergrad`, `Postgrad`, and `Research`. Each derived class inherits all the data and methods of the base class. It can also define additional data and methods or modify some of the inherited data and methods. For example, in the library information system example of Fig. 6.3, the `Library Members` base class might define the data for name, address, and library membership number for each member and its derived classes might define additional data such as max-number-books and max-duration-of-issue and which may vary for different member categories. The inheritance relationship has been represented in Fig. 6.3 using a directed arrow drawn from a derived class to its base class.

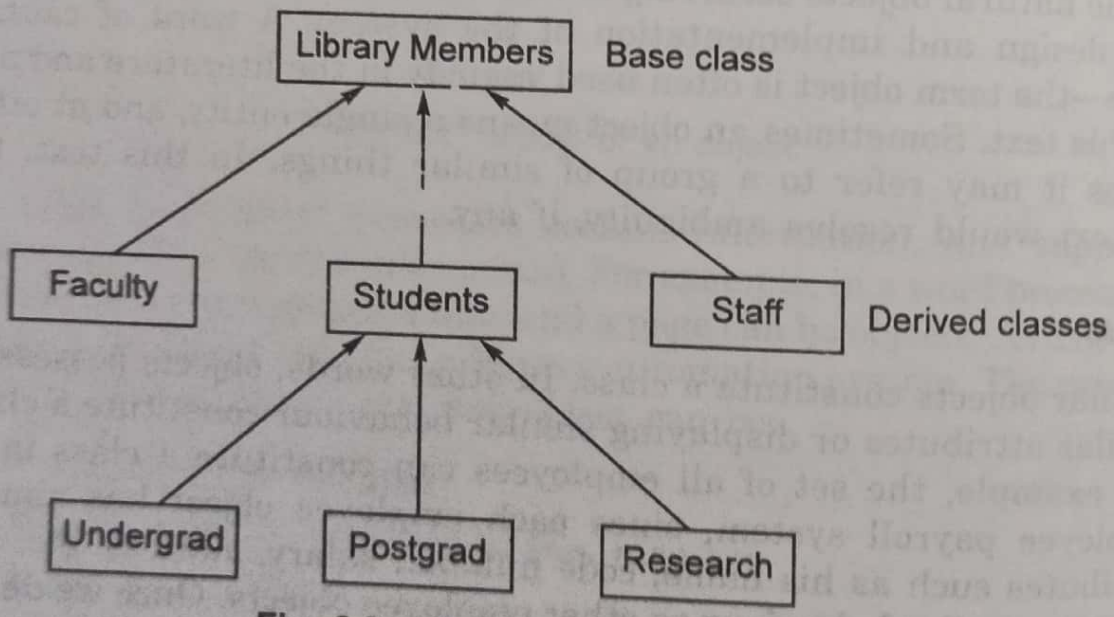


**Fig. 6.3** Library information system.

A base class is a generalization of its derived classes, i.e. the base class contains only those properties that are common to all the derived classes. In other words, the derived classes are specializations of their respective base classes in the sense that each derived class modifies or extends the basic properties of the base class in certain ways. Thus, the inheritance relationship can also be viewed as a generalization-specialization relationship. Using the inheritance relationship, the different classes of a system can be arranged in a class hierarchy (or class tree).

independent classes. ...ing the number of

## Multiple Inheritance

Multiple inheritance is the mechanism by which a subclass can inherit attributes and methods from more than one superclass. For example, in some institutions, the research students can also be staff of the institute and therefore some of the characteristics of the `Research` class might be similar to the `Student` class and some other characteristics might be similar to the `Staff` class. We can represent such a class hierarchy as in Fig 6.4 where multiple inheritance is represented by arrows directed from the subclass to the respective base classes. Observe that the class `Research` inherits from both the classes **Students** and **Staff.**

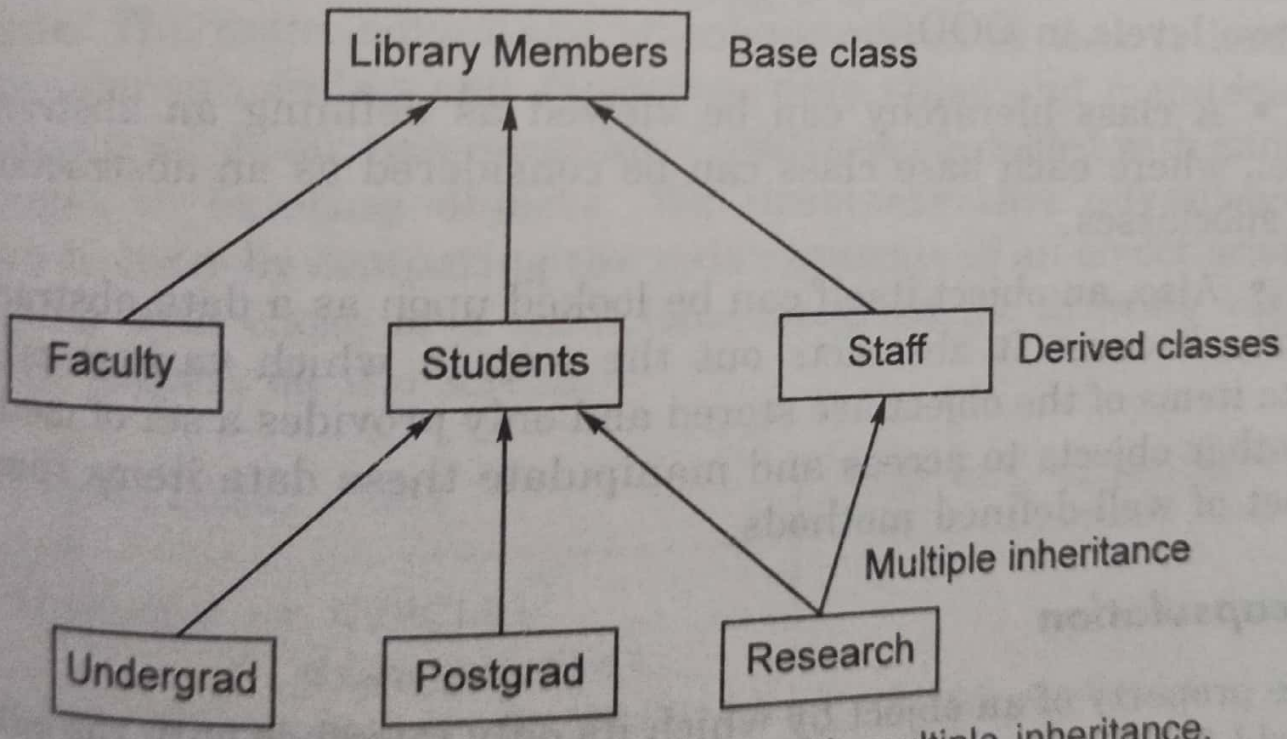


**Fig. 6.4** Library information system with multiple inheritance.

## Abstract Classes

Classes that are not intended to produce instances of themselves are called abstract classes. Abstract classes merely exist so that behaviour common to a variety of classes can be factored into one common location, where they can be defined once. Abstract classes usually

### 6.1.2 Key Concepts

#### Abstraction

Abstraction is a powerful mechanism for reducing complexity of software. In fact, it has been shown from data collected from several software development projects that software productivity is inversely proportional to software complexity. Therefore, abstraction can be viewed as a way of increasing software productivity.

Now let us examine how exactly does the abstraction mechanism work. Abstraction is the selective examination of certain aspects of a problem while ignoring other aspects of the problem. In other words, the main idea behind abstraction is to consider only those aspects of the problem that are relevant for a certain purpose and to suppress all other aspects of the problem that are not relevant for the given purpose. Thus, the abstraction mechanism allows us to represent a problem in a simpler way by omitting unimportant details. Many different abstractions of the same problem are possible depending on the purpose for which they are made. Abstraction not only helps the development engineers to understand and appreciate the problem better, but also leads to better comprehension of the system design by the end-users and maintenance teams. Abstraction is supported at two levels in OOD:

- A class hierarchy can be viewed as defining an abstraction level, where each base class can be considered as an abstraction of its subclasses.

- Also, an object itself can be looked upon as a data abstraction entity, because it abstracts out the way in which various private data items of the object are stored and only provides a set of methods to other objects to access and manipulate these data items through a set of well-defined methods.

#### Encapsulation

The property of an object by which its only interface with the outside world is by means of messages is referred to as *encapsulation*. The data of an object are encapsulated and are available only through message-based communication. This property offers three advantages:

- It protects variables of an object from corruption by other objects. This protection includes protection from unauthorized access and protection from types of problems that arise from concurrent access such as deadlock and inconsistent values.

## Polymorphism

Polymorphism literally means poly (many) morphism (f
speaking, polymorphism denotes the following:

- The same message can result in different actions
by objects of different types. This is also referred to as

- When different objects are referred to through a
in case when a message is sent, an appropriate meth
depending upon the object the pointer is currently poi
is referred to as *dynamic binding*.

Using polymorphism

## 6.4 OBJECT-ORIENTED DESIGN METHODOLOGY

Till now, very limited progress has occurred towards developing mature object-oriented design and analysis methodologies. The current object-oriented design methodologies are subjective and informal. These methodologies suggest a few guidelines, leaving the exact application of the design procedure largely to the discretion of the designer. Therefore, even while using the same design methodology, different designers can come up with different object-oriented designs, some of which might be more acceptable than the others.

### 6.4.1 A Generic Object-Oriented Development Paradigm

Instead of studying any specific object-oriented development approach, we will discuss a generic approach. Identification of objects and their attributes is the first step in any object-oriented design approach. The later steps emphasize abstraction. The ultimate goal of object-oriented system design is to locate as many methods and as much data as possible at the highest level of abstraction. The higher a method is stored in the class hierarchy, the more subclasses can share it, and the greater will be the advantage of code and design reuse. Our generic object-oriented program development paradigm consists of the following main steps:

1. Develop an informal solution strategy by studying the problem description. This informal solution strategy is also called the *processing narrative*. The problem description together with the processing narrative is often called the *extended problem description*.

2. Identify the different objects and functions (methods) needed to implement the system by analyzing the extended problem description. Although several approaches exist to determine the objects and functions of a system, a popular method of identifying the objects and the methods in the system is by examining the nouns and the verbs respectively occurring in the extended problem description.

3. Map the methods identified in Step 2 to appropriate objects. The methods of an object define its behaviour since actions can be performed on the object or by the object by invoking its methods. Therefore, it is necessary that the methods mapped to an object conform to the intuitive idea about the behaviour of the object. If necessary, the particular sequence in which messages must be sent to an object must also be identified.

4. Identify the data associated with each of the identified objects. The private data of an object can be determined by examining the data that different methods of the object need to produce their results.

5. Determine the class hierarchies. It may be necessary to define several abstract classes in order to push the methods as high as possible in the class hierarchy.

6. Identify other relationships existing among objects, such as reference, composition, and association.

7. Determine how the client classes will use the services of the server classes, i.e. specify object interfaces. In order to specify object interfaces, message structures for each object (i.e. object protocols) should be defined.

8. Represent the design using a popular graphical notation such as the one proposed by Rumbaugh and Blaha [1991].

9. Implement all objects. In order to implement an object we must implement all the data and methods identified for that object.

10. Each object is debugged separately and independently by sending test messages to it and examining the actions performed by it.

11. All objects of the system are integrated and the system is tested against its requirements (system testing).