OBJECT ORIENTED PROGRAMMING USING JAVA (CST 205)

Prepared by ***Prof.Renetha J.B. LMCST***

MODULE 1

Introduction

Topic: **Object Modeling Using Unified Modeling Language (UML)- Basic object oriented concepts**

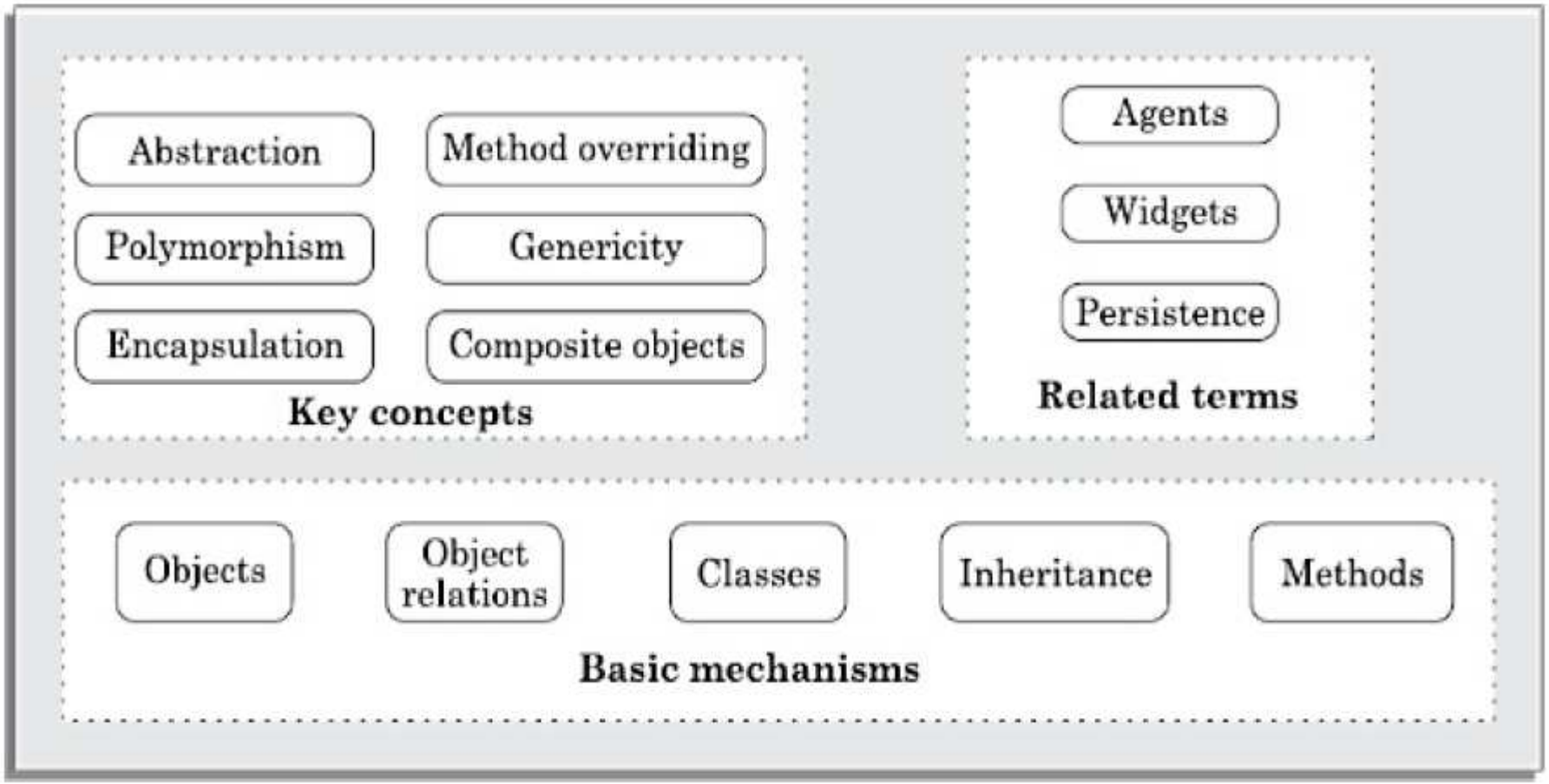
Model

- A model is constructed by **focusing only on a few aspects of the problem** and ignoring the rest.
- The model of a ***problem*** is called an ***analysis model.***
- The model of the ***solution*** (code) is called the ***design model***.
  - The design model is usually obtained by carrying out iterative refinements to the analysis model using a design methodology.

Modelling language:

- Modelling language: A modelling language consists of a **set of notations** using which design and analysis models are documented.
- A model can be documented using a modelling language such as Unified Modelling Language (UML).

BASIC OBJECT-ORIENTED CONCEPTS



OBJECT-ORIENTATION CONCEPTS

- Object
- Class
- Abstraction
- Encapsulation
- Class relationships
  - Inheritance
  - Association and link
  - Aggregation and composition
  - Dependency
- Polymorphism

Object

- Object in an object-oriented program usually represents
  - a *tangible* **real-world entity** *(can be touched)*
    - E.g.student, library member, a book, an issue register, etc.
  - or *conceptual* real-world entity
    - E,g. Loan, Job etc

Objects are real-world entities that has their own properties and behavior.

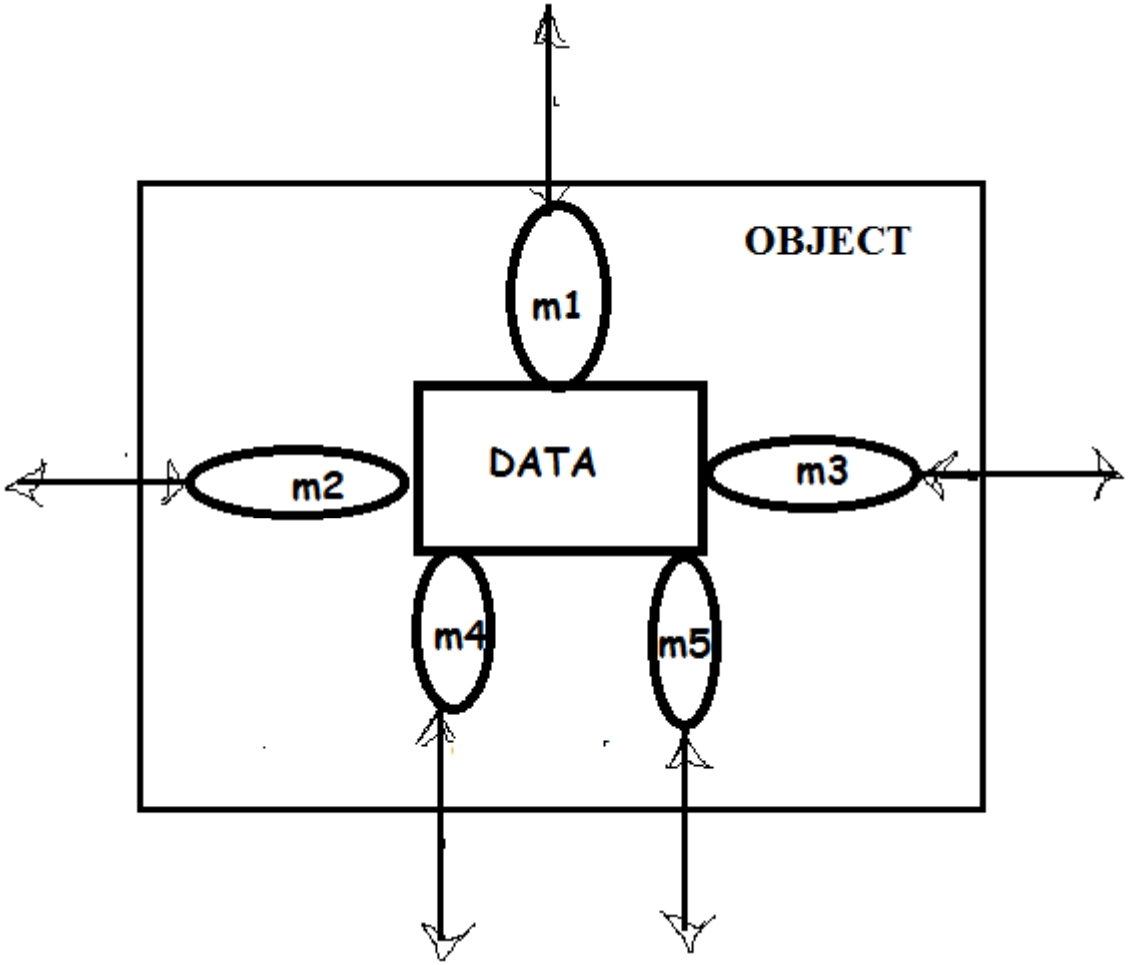
Object- Characteristics

- Each object essentially consists of
  - some **data** that is ***private*** to the object and
  - a set of **functions** (termed as operations or **methods**) that operate on those data.
- Each object hides its internal data from other objects.
- An object can access the private data of another object by invoking the methods supported by that object.

Object-Example

- Consider **Library Automation System**.
  - Objects can be library member, book, staff etc.
- Library Member object
  - private data
    - name of the member
    - membership number
    - address
  - methods
    - issue-book()
    - find-books-outstanding()
    - return-book()

Object



Here m1 , m2 etc are methods associated with object

Class

- Class is a **group of similar objects.**
- A class is a blueprint or prototype from which objects are created.
- A class is a generalized description of an object.
- An object is an instance of a class.

Class(contd.)

- All the objects in a class possess similar **attributes** (properties) and **methods** (behaviour or operation).

E.g Set of all students(objects) form **Student class**

Each student object possesses

- attributes(data)- Roll number, name etc.
- behaviour(methods)- study(), read(), write() etc.

Example

- E.g. Set of all library members(objects) would constitute the class LibraryMember
- Each LibraryMember object has
  - member name, membership number, member address, etc.------> attributes(data)
  - issue-book(), return-book(), etc.

    ----> behaviour(methods)

ADT

- An **Abstract Data Type** is a **type** with associated operations, but its *representation(inner details) is hidden*.
- ADT is based on three concepts
  - **Abstract data**- data is hidden from outside. It can be only accessed through its methods.
  - **Data structure-c**onstructed from a collection of primitive data items.
  - **Data type-**data type can be instantiated to create a variable of that type. E.g.int c;

Class-ADT

Class is an ADT

- it has abstract data
- it has structure
- we can instantiate a class into objects.

Class and Object

- **Class** is just a **logical** definition.

- Consider Student class.
  - Let Sam is a student.
  - Sam is instance/object of the class Student

  Object has a **physical existence**

Methods

- *Function inside the class is called* ***method.***
- The operations (such as create(), issue(), return(), etc.) supported by an object are implemented in the form of methods inside the class.

Difference between operation and method

- An **operation** is a specific responsibility of a class.
- The responsibility is ***implemented*** using a **method**..

Method overloading

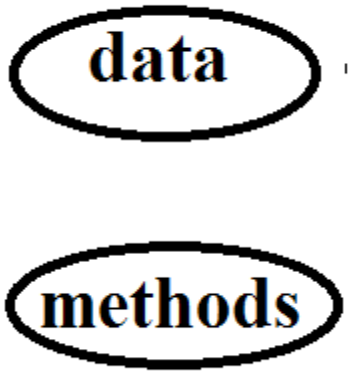
- In some cases the responsibility of a class can be implemented through **multiple methods** with the *same method name*. This is called **method overloading.**

Method overloading-Example

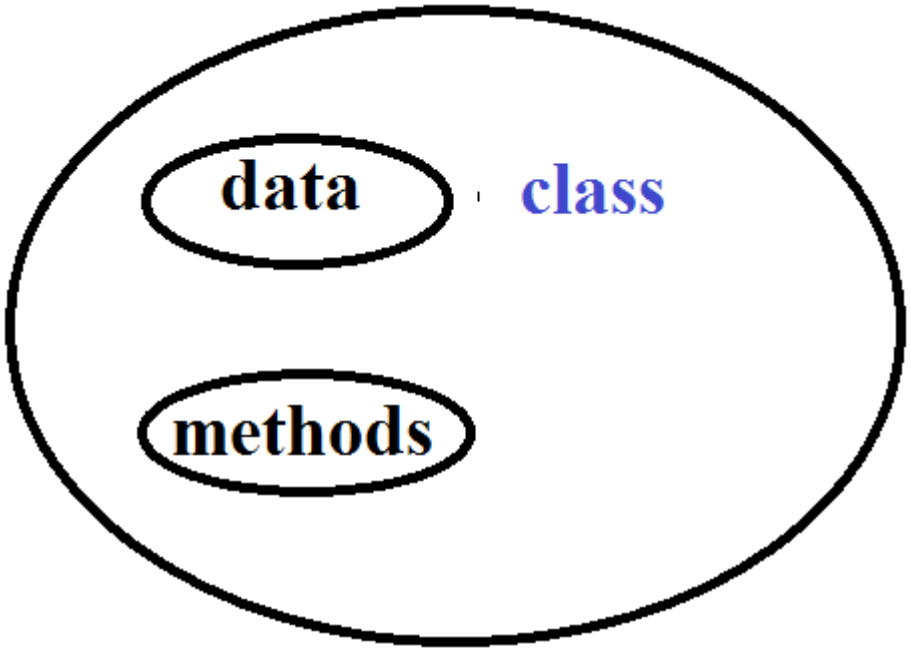
- E.g. Consider class named Circle
  - Assume that this class has three definitions for the create operation(method)—
    - int create()
      - *draws a circle with radius given inside create() function*
    - int create(int radius)
      - *draws a circle with radius passed to create() function*
    - int create(float x, float y, int radius);
      - *draws a circle with radius at specific position in xy coordinate passed to create() function*

Encapsulation

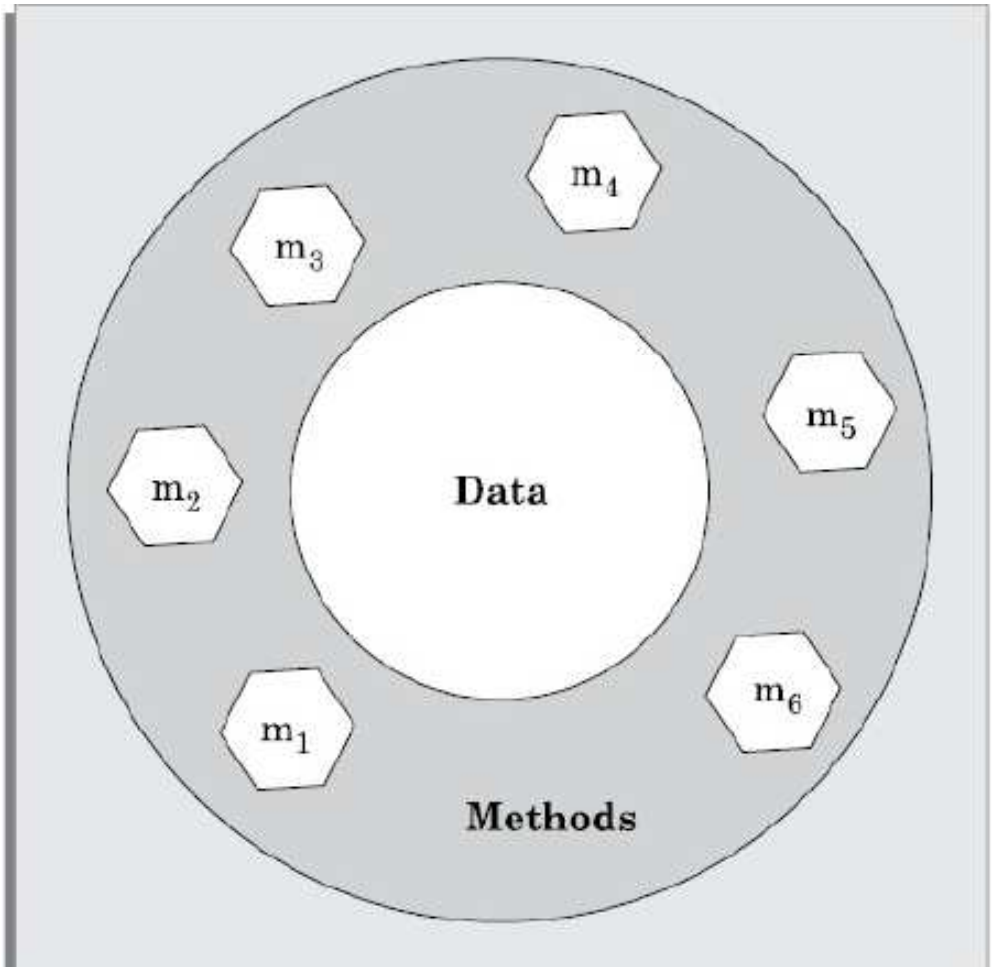
- The wrapping up of data(variables) and function (methods) into a single unit (called class) is known as encapsulation.


data
methods

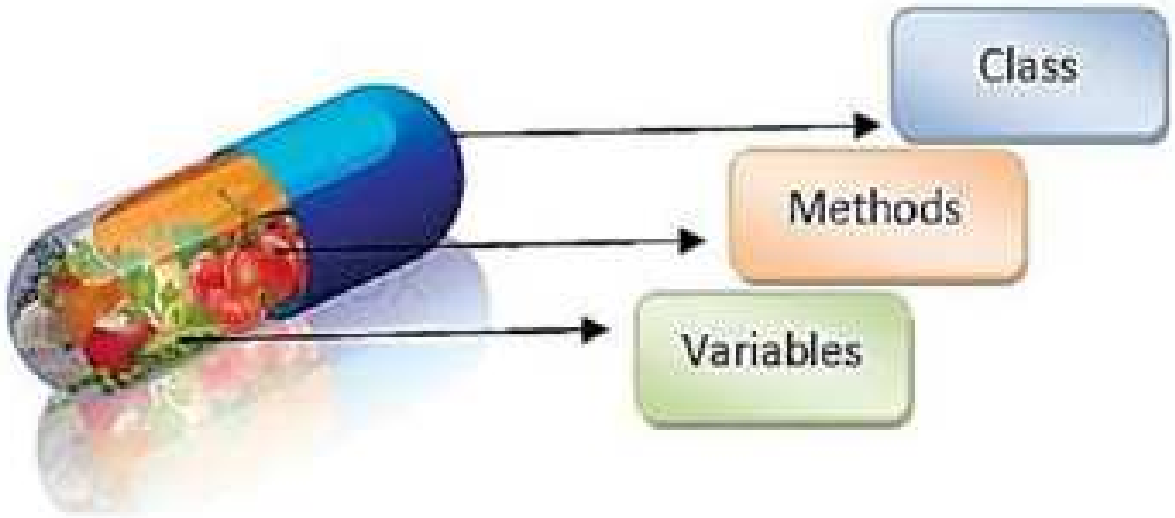
Encapsulation



Encapsulation



Encapsulation



Encapsulation - Advantage

Encapsulation offers the following three important advantages:

- **Protection from unauthorised data access:**
  - Protect data from accidental corruption and concurrent access(simultaneous access) problems.
- **Data hiding-**Helps to hide the internal structure data of an object.
  - provides abstraction, easier maintenance and bug correction.
- **Weak coupling**- Since objects do not directly change each others internal data, they are weakly coupled. This enhances understandability of the design.

Abstraction

- Abstraction mechanism
  - **consider only those aspects of the problem that are relevant** to a given purpose
  - and to suppress all aspects of the problem that are not relevant.
- Abstraction means displaying only essential information and **hiding** the inner details.

Abstraction(contd.)

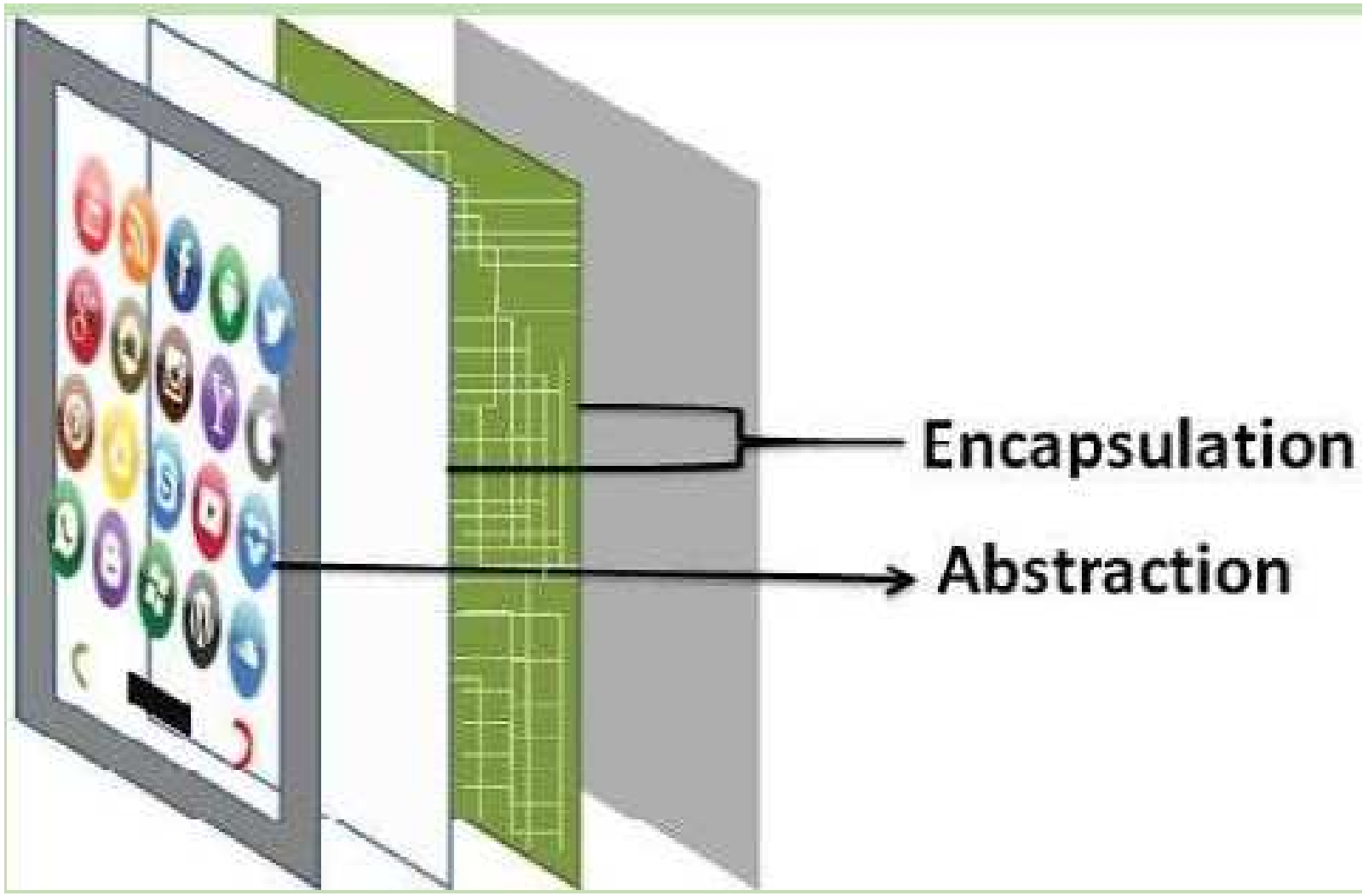
- Abstraction is supported in two different ways in an object-oriented designs (OODs).
  - **Feature abstraction-** A class hierarchy can be viewed as defining several levels (hierarchy) of abstraction, where each class is an abstraction of its subclasses.- Inheritance provides this.
  - **Data abstraction -** each object **hides** the exact way in which it **stores its internal information** from other objects .

Data abstraction-Real world example

- Consider a man driving a car. ***The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car***
- But he ***does not know about how*** on pressing accelerator the speed is actually increasing,
- *He does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car.*
- This is what abstraction is.

Data abstraction -Advantage

- An important advantage of the principle of data abstraction is that
  - It reduces coupling among various objects
    - Objects do not directly access any data belonging to each other.
  - It leads to a reduction of the overall complexity of a design.
  - It helps in easy maintenance and code reuse.


Encapsulation
Abstraction

Class Relationships

Classes in a programming solution can be related to each other in the following four ways:

- Inheritance
- Association and link
- Aggregation and composition

Inheritance

- The capability of a class to derive properties and characteristics from another class is called Inheritance.
- Inheritance is the process by which objects of one class acquired the properties of objects of another classes

Inheritance(contd.)

- **Derived Class**: The class that inherits properties from another class is called **Subclass** or Derived Class or **child** class.
- **Super Class** : The class whose properties are inherited by subclass is called **Base Class** or Superclass or **parent**class.

  E.g. Doctor is a superclass. Surgeon and Neurologist are its subclasses

Inheritance

- A base class is said to be a **generalisation** of its derived classes.
- A base class is **specialized** into derived classes.
- This means that the base class contains properties (i.e., data and methods) that are common to all its derived classes.
- Derived class inherit the properties of base class. Derived can have their special own properties also.

Base class- Derived class

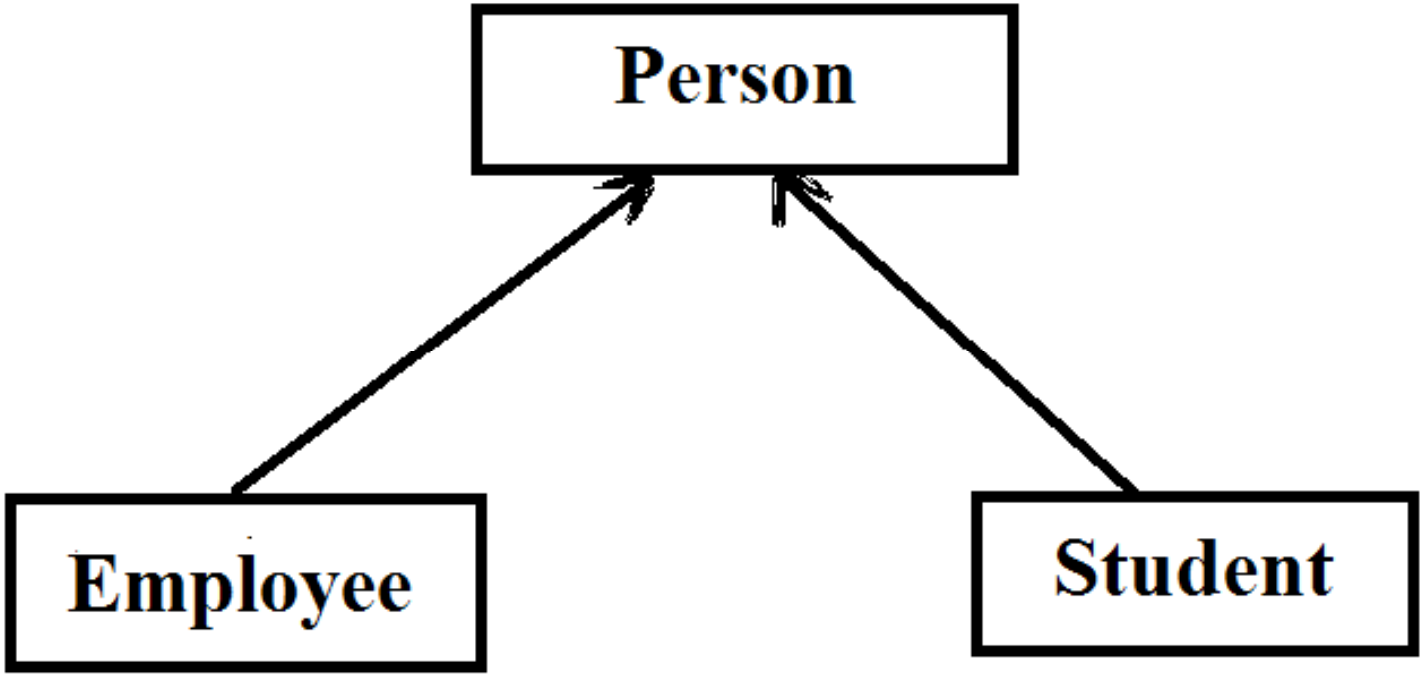
- Each derived class can be considered as a specialisation of its base class
  - because it modifies or extends the basic properties of the base class in certain ways.
- Therefore, the inheritance relationship can be viewed as a **generalisation-specialisation** relationship.

Inheritance - Example

- Consider the classes Person, Employee, Student.
  - Employee is a person
  - Student is a person
  - Employee and Student inherit the properties of Person class

Inheritance - Example

- Consider the classes Person, Employee, Student.
  - Employee is a person
  - Student is a person
  - Employee and Student inherit the properties of Person class
  - **Super(base) class** - Person
  - **Derived**(sub) class - Employee ,Student


Person
Employee
Student

Inheritance example

- Suppose Person class stores
  - ***data*** - name, aadhar number, address and data-of-birth
  - ***Methods***-enter_details(), modify_details().
- Employee is a subclass of class Person.
  - So Employee class *inherits all data and methods of Person class.*
  - It can also contain data and methods specific to employee such as also empid, designation, calculate_experience()

Inheritance example

- Suppose Person class stores
  - ***data*** - name, aadhar number, address and data-of-birth
  - ***Methods***-enter_details(), modify_details().
- Employee is a subclass of class Person.
  - contain data and methods specific to employee such as also empid, designation, calculate_experience()

Employee-

  - ***data*** - name, aadhar number, address and data-of-birth
  - ***Methods***-enter_details(), modify_details().

Inheritance - Example

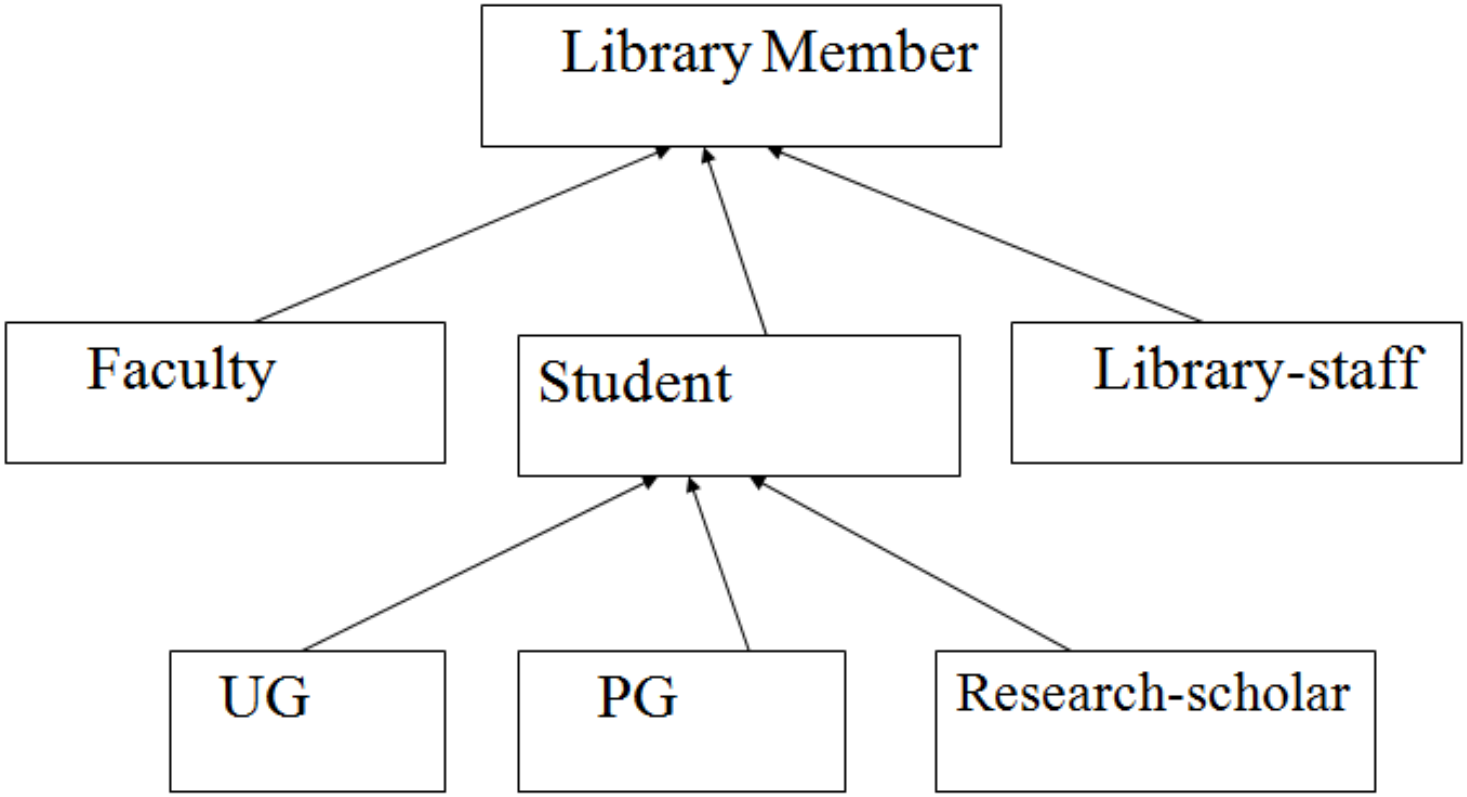
- Faculty, students, and library staff are library members.
- Base class(superclass)- ?
- Derived class(subclass) -?

Inheritance - Example

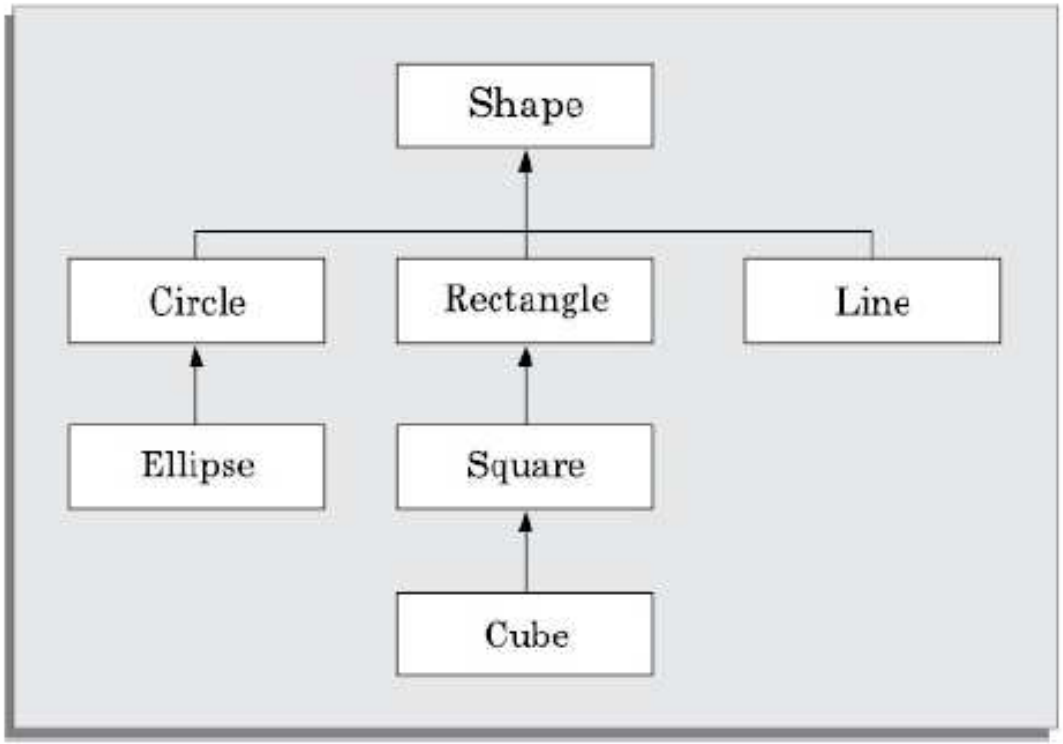
- Faculty, students, and library staff are library members.
- Base class(superclass)- Library member
- Derived class(subclass) -Faculty ,students , library staff

Inheritance - Example

- Faculty, students, and library staff are library members. Students fall in three categories PG, UG and research-scholars.



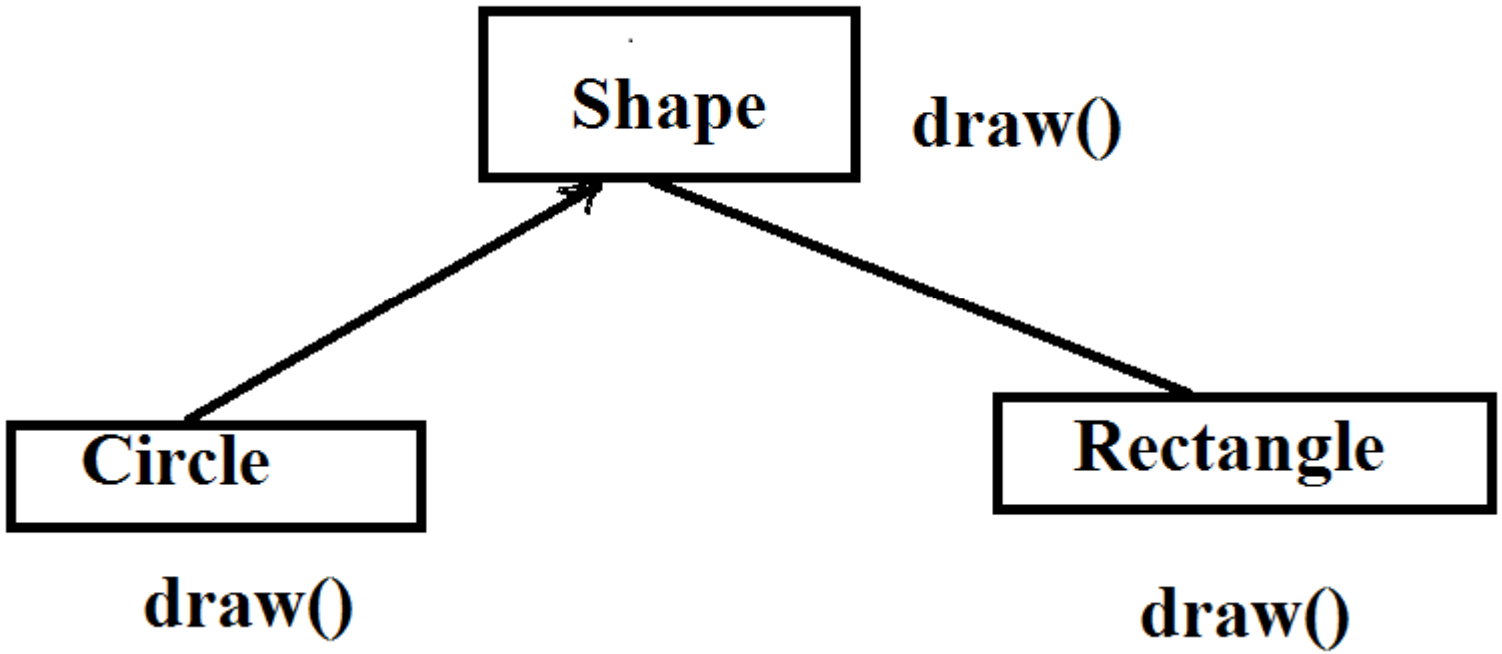
Class hierarchy



Class hierarchy of geometric objects

Method overriding

- When a method in the base class is also defined in a derived class, then the method is said to be overridden in the derived class.
- If subclass contain same method name as superclass then subclass method overrides superclass method

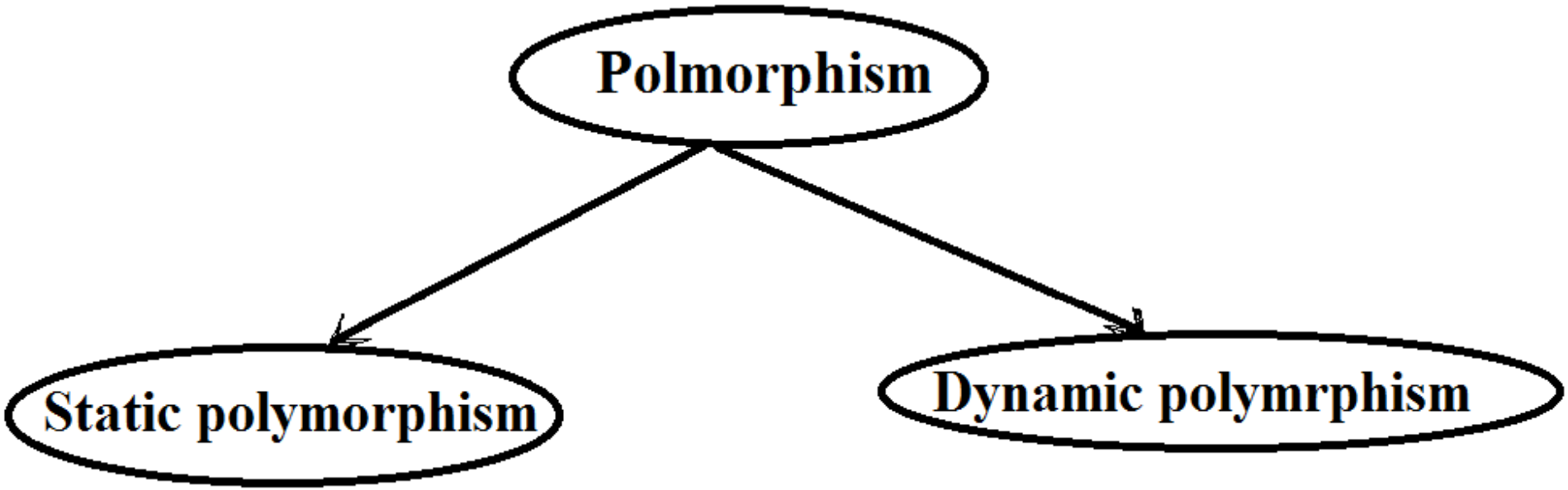


Method overriding

Method overriding

Polymorphism

- Polymorphism literally means poly (many) morphism (forms).
- Real world example
  - Diamond, graphite, and coal are called polymorphic forms of carbon


Polmorphism
Static polymorphism
Dynamic polymrphism

Static polymorphism

- Static polymorphism occurs when multiple methods implement the same operation.
- Static polymorphism is also called **static binding**.
- If a program has *different methods* *with* ***same name*** *but different parameter types*
  - when that method is invoked(called), (there are different methods with same name here), the exact method which is to be bound to the method-call is determined at ***compile-time*** (statically).

Method overloading- static binding

- **Method overloading** is a static polymorphism. Here a program (class) can have ***different functions with same name*** (but has different parameters).

Example-static polymorphism

Suppose class **Circle** has so many data and methods. Let Circle class has three methods named *create*

int create()

int create(int radius)

int create(int x, int y, int radius)

This is an example for **method overloading.**

Example-static polymorphism(contd.)

✓When *create function is called without parameter* then **int create()** method is invoked.

✓When *create function is called with one integer parameter* then **int create**(int radius) method is invoked.

✓When *create function is called with three integer parameters* then **int create**(int x, int y, int radius) method is invoked.

Dynamic polymorphism

- Dynamic polymorphism is also called **dynamic binding**.
- In dynamic binding, when a method is called , the exact method to be invoked (bound) is known at the **run time (dynamically)** and *cannot be determined at compile time.*

Dynamic polymorphism(contd.)

- Dynamic binding is based on two important concepts:
  - Assignment of an object to another compatible object.
  - Method overriding in a class hierarchy.

Dynamic polymorphism(contd.)

- **Assignment to compatible of objects**
  - In object-orientation, ***objects of the derived classes are compatible with the objects of the base class***.
  - That is, an object of the derived class can be assigned to an object of the base class, but not vice versa.
- **Method overriding**
  - If subclass contain same method name as superclass then subclass method overrides superclass method

Dynamic binding summary

- Even when the method of an object of the base class is invoked,
  - an appropriate overridden method of a derived class would be invoked
    - depending on the exact object that may have been assigned at the run-time to the object of the base class.

Advantage of dynamic binding

- The principal advantage of dynamic binding is that
  - it leads to **elegant programming** and facilitates **code reuse and maintenance**.
  - code is much more concise, understandable, and intellectually appealing

Association

- Association is a common type of relation among classes.
- The association relationship can either be ***bidirectional*** or ***unidirectional***.
- An association describes a ***group of similar links.***
  - A link can be considered as an instance of an association relation.
- An association between two classes simply means that *zero or more links may be present among the objects of the associated classes at any time during execution.*

Association

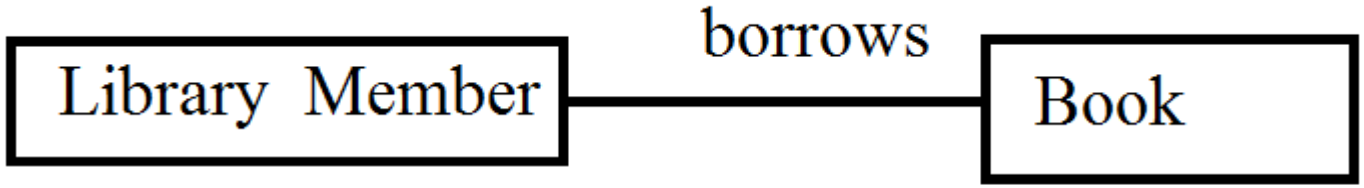
- When two classes are associated, they can take each others help (i.e. invoke each others methods) to serve user requests.(**binary association**)
  - if one class is associated with another *bidirectionally*, then the corresponding objects of the two classes know each others ids(identities).

Association-Example

- A Student can register in one Elective subject.
  - Here, the class Student is associated with the class ElectiveSubject.
  - Therefore, an ElectiveSubject object would
    - *know the ids* of all Student objects that have registered for the Subject
    - and *can invoke their methods* such as printName, printRoll a nd enterGrade.
- This is example for binary association

Association-Example

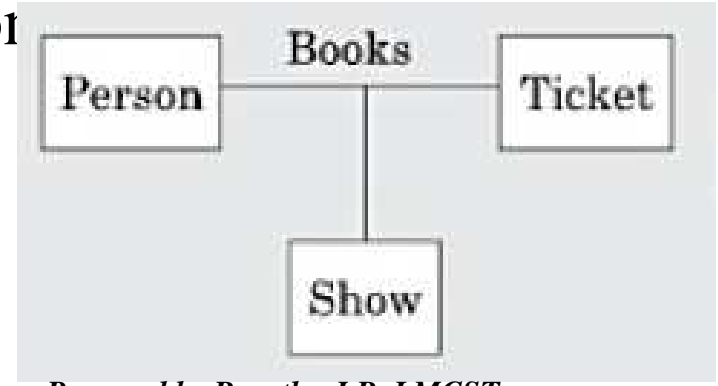
- Consider another example of association between two classes: Library Member borrows Books.



  - Here, ***borrows*** is the association between the class **LibraryMember** and the class **Book**.
  - The association relation would imply that given a book, it would be possible to determine the borrower and vice versa.

Association(contd.)

- **n-ary association**
  - Three or more different classes can be involved in an association.
  - If three classes are associated then it is called 3-ary (ternary) association
  - E.g. A person books a ticket for a certain show.
    - Here, an association exists among the classes Person, Ticket, and Show. This is ternary associatio



Association(contd.)

- A class can have an ***association relationship with itself.*** This is called
  - **recursive association** or **unary association.**
  - Example, consider the following—two students may be friends. Here, an association named *friendship* exists among pairs of objects of the Student class

Association(contd.)

- Links are time varying (or **dynamic**) in nature.
- Association relationship between two classes is **static** in nature.
  - If two classes are associated, then the association relationship exists at all points of time.
  - But links between objects are dynamic in nature.
  - Links between the objects of the associated classes can get formed and dissolved as the program executes.

Association(contd.)

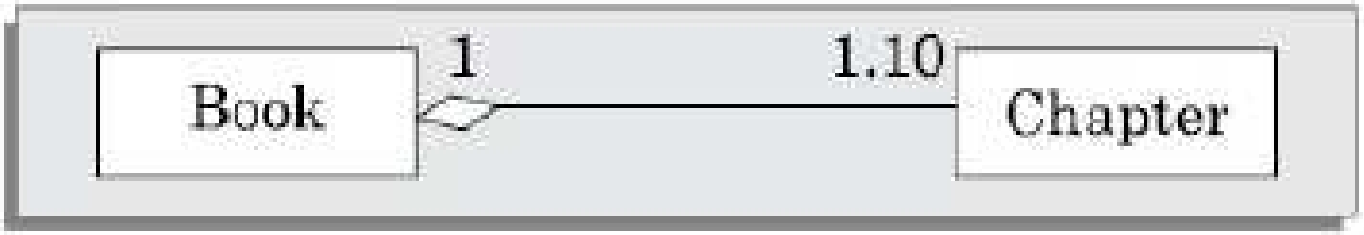
- Example, an association relationship named **works-for** exists between the classes **Person** and **Company**.
  - Ram works for Infosys,
    - This implies that a link exists between the object Ram and the object Infosys.
  - Hari works for TCS
    - A works for link exists between the objects Hari and TCS.
  - If Ram may resign from Infosys and join Wipro. In this case, the link between Ram and Infosys breaks and a link between Ram and Wipro gets formed.
  - In all these case association works for remains there

Composition and aggregation

- Composition and aggregation represent **part/whole relationships** among objects.
- Composition/aggregation relationship is also known as **has a** relationship.
- Objects which contain other objects are called composite objects.

Composition and aggregation- Example

- Example: A Book object can have upto ten Chapters.
  - Here a Book object is said to be composed of upto ten Chapter objects.
  - A Book **has** upto ten Chapter objects



Composition and aggregation(contd.)

- Aggregation/composition can occur in a hierarchy of levels.
  - That is, an object may contain another object. This latter object may itself contain some other objects.
- Composition and aggregation relationships **cannot be reflexive.**
  - That is, an object cannot contain an object of the same type as itself.

Dependency

- A class is said to be dependent on another class,
  - if any changes to the latter class requires a change to be made to the dependent class.

  E.g. class1 is dependent on class2 if any change is made in class2 then change is required in class1 too.
- A dependency relation between two classes shows that **any change made to the independent class** would require the *corresponding change to be made to the dependent class*.

Dependency(contd.)

- Two important reasons for dependency to exist between two classes are the following:
  - A ***method*** of a class takes an **object of another class as an argument**.
    - E.g.

```
class A
{
int function(class B){  ……..}
}
```

  - A **class** *implements* an **interface** class.
    - If some properties of the interface class are changed, then a change becomes necessary to the class implementing the interface class as well.

Summary of class relationship

- **Aggregation**
  - B is a part of A
  - A contains B
  - A is a collection of Bs
- **Composition**
- B is a permanent part of A
- A is made up of Bs
- A is a permanent collection of Bs

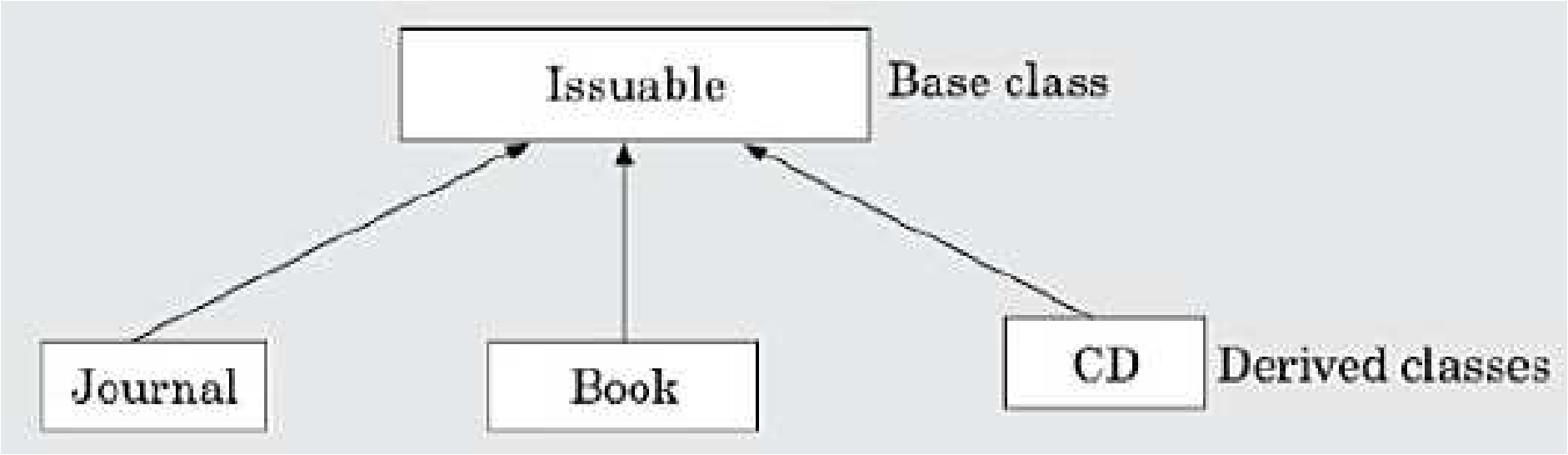
Summary of class relationship

- **Inheritance**
  - A is a kind of B
  - A is a specialisation of B
  - A behaves like B
- **Association**
  - A delegates to B
  - A needs help from B
  - A collaborates with B. Here collaborates with can be any of a large variety of collaborations that are possible among classes such as employs, credits, precedes, succeeds, teaches etc.

Abstract class

- Classes that are not intended to produce instances(objects) of themselves are called abstract classes.
  - an abstract class cannot be instantiated.
  - ***Objects cannot be created*** from abstract class.
  - Abstract class ***act as base class*** in inheritance.
- Abstract classes usually support generic methods.
- Advantage
  - code reuse can be enhanced
  - the effort required to develop software brought down.

Abstract class- Example



- Here Issuable is an abstract class and cannot be instantiated.

Advantages of OOD

- Code and design reuse
- Increased productivity
- Ease of testing and maintenance
- Better code and design understandability enabling development of
- large programs

Disadvantages of OOD

- The principles of abstraction, data hiding, inheritance, etc. do incur ***runtime overhead***.
- An important consequence of object-orientation is that the data that is centralized in a procedural implementation, gets ***scattered across various objects*** in an object-oriented implementation.

Object-oriented Programming Language(OOPL)

- The first object-oriented programming language was Smalltalk.

- Other OOPL are C++, Java etc.

Reference Text Book

- Rajib Mall, Fundamentals of Software Engineering, 4th edition, PHI, 2014.

Thank you