**Design Processes:**
- During the design phase the classes identified in OOA must be revisited with a shift in focus to their implementation. New classes or attributes & methods are to added for implementation purposes & user interfaces.
- The *object-oriented design* process consists of following activities:
  1. **Apply design axioms** to design classes, their attributes, methods, associations, structures & protocols. It constitutes two separate steps
     - Refine & complete the static UML class diagram by adding details. This steps consists of:
       - Refine attributes
       - Design methods & protocols by UML activity diagram to represent method’s algorithm
       - Refine associations between classes (if required)
     - Refine class hierarchy & design with inheritance (if required)
     - **Iterate** and refine again
  2. Design the access layer
     - **Create mirror classes:** For every business class identified & created, create one access class
     - **Identify access layer class relationships**
     - **Simplify classes & their relationships:** Main goal – eliminate redundant classes & structures
       - **Redundant classes:** Don’t keep 2 classes that performs ||| translate request & results
       - **Method classes:** Try to eliminate classes with 1 or 2 methods by combining with others
     - **Iterate & Refine** again
  3. Design the view layer classes
     - **Design the macro level** user interface, identifying view layer objects
     - **Design the micro level** user interface, which includes the following activities
       - Design view layer objects by applying the design axioms & corollaries
       - Build a prototype of view layer interface
     - **Test usability & user satisfaction**
     - **Iterate and refine**
  4. **Iterate & refine the whole design.** Reapply design axioms, if needed repeat preceding steps
- Design must be traceable across requirements, analysis, design, code & testing. There must be clear step-by-step approach to design from the requirements model. All designed components must directly trace back to user requirements.

**Design Axioms:**
- An **axiom** is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception. They are hypothesized from a large number of observations by noting the common phenomena shared by all cases
- A **theorem** is a proposition that may not be self-evident but can be proven from accepted axioms
- A **corollary** is a proposition that follows from an axiom or another proposition that has been proven
- As per Suh’s design axioms to OOD, two axioms are followed
  - **Axiom 1:** Independence Axiom. Maintain the independence of components
  - **Axiom 2:** Information Axiom. Minimize the information content of the design
  - **Axiom 1** states that, during the design process, as we go from requirement and use case to a system component, each component must satisfy that requirement without affecting other requirement
  - **Axiom 2** is concerned with simplicity. Occam razor says that “The best theory explains known facts with a minimum among of complexity and maximum simplicity & straightforwardness”

**Corollaries:**
- Many corollaries may be derived as a direct consequence of axioms. These corollaries may be more useful in making specific design decisions, since they can be applied to actual situations more easily than original axioms. They are also called design rules and are derived from 2 basic axioms
  - **Corollary 1:** Uncoupled design with less information content.
  - **Highly cohesive objects** can improve coupling because only a minimal amount of essential information need be passed between objects.
The main goal here is to maximize objects cohesiveness among objects & s/w components in order to improve coupling because only a minimal amount of essential information need be passed between components.

Coupling is a measure of strength of association established by a connection from one object or s/w component to one another.

The degree or strength of coupling between two components is measured by amount & complexity of information transmitted between them. Coupling increases (becomes stronger) with increasing complexity or obscurity of the interface.

Interaction coupling involves amount & complexity of messages between components. It is desirable to have little interaction.

Inheritance is a form of coupling between super and sub classes. A sub class is coupled to its superclass in terms of attributes and methods.

Cohesion reflects the “single–purposeness” of an object. Method cohesion function cohesion means that a method should carry only one function. A method that carries multiple functions is undesirable. Class cohesion means that all the class’s methods & attributes must be highly cohesive meaning to be used by internal methods or derived classes’ methods.

Corollary 2: Single Purpose. Each class must have a single, clearly defined purpose. When we document, we should be able to easily describe the purpose of a class in a few sentences.

Corollary 3: Large number of simple classes. Keeping the classes simple allows reusability.

The smaller are classes, the better are chances of reusing them in other projects. Large & complex classes are too specialized to be reused.

The primary benefit of s/w reusability is higher productivity. The s/w development team that achieves 80% reusability is 4 times as productive as team that achieves 20% reusability.

Corollary 4: Strong mapping. There must be a string association between the physical system (analysis’s object) & logical design (design’s object).

Corollary 5: Standardization. Promote standardization by designing inter changeable components and reusing existing classes or components.

Corollary 6: Design with inheritance. Common behavior (methods) must be moved ro super classes. The superclass–subclass structure must make logical sense.

Design Patterns:

Design patterns are devices that allow systems to share knowledge about their design. These describe commonly recurring problems.

A perfect design pattern has the following documentation concepts.

Pattern Name: Façade
Rationale & Motivation: The façade pattern can make the task of accessing a large number of modules much simpler by providing an additional interface layer. Classes: In façade system at least 4 or more classes are required: client, façade & classes underneath façade. In typical situation, façade would have a limited amount of actual code, making calls to lower layers most of time.

Advantages/ disadvantages: The primary advantaged is making interfacing between many modules or classes more manageable. One possible disadvantage is that we may lose some functionality contained in lower level of classes which depends on how the façade designed.

Examples: Imagine a client request program a room that is used to view objects all in room. Using a design pattern façade eliminates the need for the client class to deal with a large number of classes.

```
Client
  ↓
Façade
  ↓
Projector  Screen  Window
  ↓
Something
```
Designing Classes:
- The single most important activity in designing an application is coming up with a set of classes that works together to provide the needed functionality. Underlying the functionality of any application is the quality of its design.

OOD Philosophy:
- A great benefit of OO approach is that classes organize related properties into units that stand on their own. Applying design axioms and carefully designed classes can have a synergistic effect, not only on the current system but on its future evolution.

UML – OCL:
- UML is a graphical language with a set of rules & semantics in English in form of OCL. Object Constraint Language (OCL) is specification language that uses simple logic for specifying properties of a system. Syntax of some common navigational expressions is shown here:
  - `item.selector`: The selector is name of an attribute in item. The result is the value of attribute.
  - `item.selector[qualifier-value]`: The selector indicates a qualified association that qualifies the item. Result is related object selected by qualifier, eg., array indexing as form of qualification.
  - `SetÆselect(Boolean-expression)`: Boolean expression is written in terms of objects within set.

Designing Classes – The Process:
- It constitutes step 1 of OOD process is an iterative process and at each iteration, the design has been improved.

Class visibility:
- The main objective is designing well defined public, private & protected protocols.
- Public protocols define the functionality & external messages of an object; private protocols define implementation of an object.
- Private Protocol (visibility) of class includes messages that normally should not be sent from other objects. In protected protocol (visibility), subclasses use method in addition to class itself.
- The problem of encapsulation leakage occurs when details about a class’s internal implementation are disclosed through the interface.

Refining Attributes:
- Attributes identified in OOA must be refined on implementation during this phase. In design phase, detailed information must be added to the model. The main goal is to refine existing attributes or add attributes that can elevate the system into implementation.
- Attribute Types:
  - Three types of attributes are 1. Single-value attributes 2. Multiplicity or multivalue attributes and 3. Reference to another object, or instance connection.

Designing Methods and Protocols:
- The main goal of this activity is to specify the algorithm for methods identified so far. A class can provide several types of methods: Constructor, Destructor, Conversion method, Copy method, Attribute set, Attribute get, I/O methods, Domain specific.

Using private and protected protocols to define the functionality of the object. Remember five rules to avoid bad design:
1. If it looks messy, then its probably a bad design.
2. If it looks too complex, then its probably a bad design.
3. If it is too big, then its probably a bad design.
4. If people don’t like it, then its probably a bad design.
5. If it doesn’t work, then its probably a bad design.

Apply design axioms and corollaries to avoid design pitfalls and use UML operation presentation which is similar to syntax of UML attribute representation.

Packages and Managing Classes:
- The UML package is grouping of model elements. It can organize the modeling elements including classes. Packages themselves may be nested within other packages.
- A package may contain both other packages and ordinary model elements. The entire system description can be thought of as a single, high-level sub system package with everything else in it.
- OOD is an iterative process. Designing is as much about discovery as construction. At each iteration we can improve the design.

The advisable trick is to fix the design as early as possible; redesigning late in development cycle is problematic and may be impossible.
Object Storage:
- A database management system (DBMS) is a set of programs that enables the creation & maintenance of a collection of related data. The fundamental purpose is to provide a reliable, persistent data storage facility & mechanisms for efficient, convenient data access & retrieval.
- **Persistence** refers to the ability of some objects to outlive the programs that created them. A program will create a large amount of data throughout its execution. Each item of data will have a different life time. Atkinson et al. describe **six broad categories** of life time of data:
  1. Transient results to evaluation of expressions
  2. Variables involved in procedure activation (parameters & variables with a localized scope)
  3. Global variables & variables that are dynamically allocated
  4. Data that exist between the executions of a program
  5. Data that exist between the versions of a program
  6. Data that outlive a program
- The first three categories are **transient data**, data that cease to exist beyond lifetime of creating process. The other three are non-transient or persistent data.
- A file or a database can provide a longer life for objects – longer than duration of process in which they were created. From a language perspective, this characteristic is called persistence. Essential elements in providing a persistent store are:
  - Identification of persistent objects or reachability (**object ID**)
  - Properties of objects & their interconnections. The store must be able to coherently manage non-pointer & pointer data (i.e., **inter-object references**)
  - Scale of the object store. The object store should provide a conceptually infinite store

Stability: The system should be able to recover from unexpected failures and return the system to a recent self–consistent state. This is similar to reliability requirements of a DBMS.

Database Management Systems:
- A DBMS is a set of programs that enable the creation & maintenance of a collection of related data. They have number of properties that distinguish them from file–based data management approach.
- A **fundamental characteristic** of database approach is that DBMS contains not only data but complete definition of data formats it manages. This description is known as schema or meta–data containing a complete definition of data formats, such as data structures, types & constraints.
- **Advantage** of database approach is that it will provide a generic storage management mechanism. Another one is program–data independence.
- **Database Views**: DBMS provides the database users with a conceptual representation that is independent of low–level details (physical view) of how the data are stored. The database an provide an abstract data model that uses logical concepts such as field, records, tables & their interrelationships. DBMS can provide multiple virtual views of data that are tailored to individual applications. This allows convenience of private data representation with advantage of globally managed information.
- **Database Models**: It is a collection of logical constructs used to represent the data structure & data relationships within the database. It is grouped into two categories:
  - **Conceptual model** is concerned with what is represented in the database & Implementation model is concerned with how it is represented. It can be stated as **Hierarchical model**, Network model, Relational model (**tuples – Primary key & foreign key**).
- **Database Interface**: The interface on a database must include a data definition language (DDL), query and data manipulation language (DML).
  - Database systems adopt **two approaches** for interfaces with system. One is to embed a database language (SQL) & other is to extend the host programming language with database constructs.
  - **Database Schema & DDL**: DDL is the language used to describe structure of & relationships between objects stored in a database. This structure of information is database schema.
  - **DML & Query Capabilities**: DML is language that allows users to access & manipulate data organization. SQL is standard DML for relational DBMSs. It is widely used for its query capabilities. The query usually specifies:
    - The domain of the discourse over which to ask the query
    - The elements of general interest
    - The conditions or constraints that apply
    - The ordering, sorting or grouping of elements & constraints that apply to the ordering or grouping

DML are either procedural or nonprocedural. A procedural DML requires users to specify what data are desired & how to get data. Non procedural DML requires to specify what data are needed not how.
Logical & Physical Database Organization & Access Control:
- Logical database organization refers to conceptual view of database structure & relationships within the database. Physical database organization refers to how logical components of database are represented in a physical form by operating system constructs.

- Share ability & Transactions: Data & objects in database often need to be accessed & shared by different applications. With multiple applications having access to object concurrently, database must detect & mediate conflicts & promote greatest amount of sharing possible without sacrificing integrity of data. A transaction is a unit of change in which many individual modifications are aggregated into a single modification that occurs in its entirety or not at all. A transaction is said to commit if all changes can be made successfully to database and to abort if cancelled. This ability of transactions ensures atomicity of change that maintain the database in a consistent state.

- Concurrency Policy: The most conservative way to enforce serialization is to allow a user to lock all objects of records when they are accessed and to release the locks only after a transaction commits. This approach, traditionally known as conservative or pessimistic policy, provides exclusive access to object, despite what is done to it. Under optimistic policy, two conflicting transaction are compared in their entirety & then their serial ordering is determined.

Distributed Databases & Client –Server computing:
- Many modern databases are distributed databases. This implies that portions of database reside on different nodes (computers) & disk drives in network. Usually, each portion of database is managed by server, process responsible for controlling access & retrieval of data from DB portion.

- The server dispenses information to client applications & makes queries or data requests to these client applications or other servers. Clients generally reside on nodes in network other than those on which servers execute.

- Client–Server Computing is logical extension of modular programming. The fundamental assumption of modular programming is that separation of a large piece of s/w into its constituent parts (“modules”) creates the possibility for easier development & better maintainability.

- In two tier architecture, a client talks directly to a server, with no intervening server. This type of architecture is typically used in small environments with less than 50 users.

- Three-tier architecture introduces a server (application or web server) between client & server. The role of application or web server is manifold. It can provide translation services, metering services or intelligent agent services.

- Distributed processing implies that processing will occur on more than one processor in order for a transaction to be completed.

- Typical client–server application consists of following components:
  1. User interface: This major component of client–server application interacts with users, screens, windows, windows management, keyboard & mouse handling.
  2. Business processing: This part of application uses the user interface data to perform business tasks. This component has been developed by utilizing an object–oriented technology.
  3. Database processing: This part of application code manipulates data within application. Data manipulation is by DML such as SQL.

- Cooperative processing is computing that requires two or more distinct processors to complete a single transaction. It is form of distributed computing in which two or more distinct processes are required to complete a single business transaction.

Next Generation of Client–Server Computing:
- S/w technology is in the midst of a major computational shift toward distributed object computing (DOC). It promises the most flexible client–server systems, because it utilizes reusable s/w components that can roam anywhere in networks, run on different platforms, communicate with legacy applications by means of object wrappers and manage themselves & resources they control.

- Distributed object technology has been tied to standards from early stage. Since, 1989, Object Management Group (OMG) with over 500 members companies, has been specifying architecture for an open s/w bus on which object components written by different vendors can operate across networks & operating system.

- Several competing DOC standards–OMG’s CORBA, OpenDoc & Microsoft’s ActiveX / DCOM.
Common Object Request Broker Architecture (CORBA): It was a standard proposed as a means to integrate distributed heterogeneous business applications & data.

The CORBA interface definition language (IDL) allows developers to specify language-neutral, Object oriented interfaces for application & system components. IDL definitions are stored in an interface repository, a sort of phone book that offers object interfaces & services which is central to communication among objects located on different systems.

CORBA object request brokers (ORBs) implement a communication channel through which applications can access object interfaces and request data & services. CORBA common object environment (COE) provides system level services such as life cycle management of objects, event notification between objects & transaction and concurrency control.

Microsoft’s ActiveX / DCOM: -

Microsoft’s component object model (COM) & its successor the distributed component object model (DCOM) are Microsoft’s alternatives to OMG’s distributed object architecture CORBA. It is supported mostly by obe enterprise, Microsoft.

DCOM is an Internet & component strategy where ActiveX (formerly called Object Linking & Embedding – OLE) plays the role of DCOM object. It also is backed by a very efficient Web browser, the Microsoft Internet Explorer.

Object Oriented Database Management Systems (OODBMS): -

To meet broader class of applications with unconventional & complex data type requirements, popularity of OOP lead to concept of OODBMS. It is combination of OOP and database technology which is popularly called as object oriented databases.

The rules of OODBMS form combination of both OO language properties & DBMS properties.

The rules that make an object oriented system are as follows:
1. The system must support complex objects which combination of atomic types of objects
2. Object identity must be supported and existence independent of its values
3. Objects must be encapsulated with program & its data embodies separation of interface
4. The system must support types or classes with their individual concepts
5. The system must support inheritance whose advantage is shared codes & interfaces
6. The system must avoid premature binding also known as late binding or dynamic binding
7. The system must be computationally complete
8. The system must be extensible equal to status of system predefined types

These requirements are met by most OOP languages like C++, etc. Then comes the set of rules make a DBMS are

9. It must be persistent, able to remember an object state – data survive beyond execution time
10. It must be able to manage very large databases – 2e storage & provide performance features
11. It must accept concurrent users – support notions of atomic, serializable transactions
12. It must be able to recover from h/w & s/w failures – return to a coherent state
13. Data query must be simple – graphical browser might fulfill this requirement sufficiently.

Object Relational Systems: -

Object oriented databases has advantages such as * A large number of different data types, * A large number of relationships between the objects, * Objects with complex behaviors

Creating an object model from an existing relational database layout (schema) often is referred to as reverse engineering. Conversely, creating a relational schema from an existing object model often is referred to as forward engineering.

Object – Relation Mapping: In relational database, the schema is made up of tables, consisting of rows and columns where each column has a name & a simple data type. A tuple of table contains data for a single entity that correlates to an object in an object-oriented system. In addition, a stored procedure in a relational database may correlate to a method in an OO architecture.

A stored procedure is a module of precompiled SQL code maintained within the database that executes on server to enforce rules the business has set about the data. Therefore, the mapping essential to object & relational integration are between a table & class, between columns & attributes, between a row & an object and between a stored procedure & a method.

To define how relational data maps to and from application objects, it must have at least following mapping capabilities: * Table–class mapping * Table–multiple classes mapping * Table – inherited classes mapping * Tables–inherited classes mapping

Keys for instance navigation: In addition to simple data conversion, mapping of column values defines the interpretation of relational foreign keys. The mapping describes how foreign key can be navigate among classes, instances in mapped object model & referential integrity is maintained.
A foreign key defines a relationship between tables in a relational database. In an object model, this association is where objects can have references to other objects that enable instance-to-instance navigation. A popular mechanism in relational databases is use of stored procedures.

**Multi–Database Systems:**
- **Heterogeneous information** systems facilitate the integration of heterogeneous information sources, where they can be structured, semi-structured & some times unstructured. Such heterogeneous information systems are referred to as federated multi-database systems.
- A multi-database systems (MDBS) is a database system that resides unobtrusively on top of, say, existing relational and object databases & file systems & presents a single database illusion to users.
- The global schema is constructed by consolidating (integrating) the schemata of local databases; schematic differences (conflicts) among them are handled by neutralization (homogenization), the process of consolidating the local schemata.
- The distinctive characteristics of multi-database systems:
  - Automatic generation of a unified global database schema from local databases, in addition to schema capturing and mapping for local databases.
  - Provision of cross-database functionality (global queries, transactions) by unified schemata.
  - Integration of heterogeneous database systems with multiple databases.
  - Integration of data types other than relational data through use of tools – driver generators, etc.
  - Provision of a uniform but diverse set of interfaces (e.g. an SQL-style & C++) to access & manipulate data stored in local databases.
- Open Database Connectivity: Multi-database Application Programming interfaces: The benefits of being able to port database applications by writing to an API for a virtual DBMS are so appealing to s/w developers that computer industry recently introduced several multi-database API’s.

**ODBC** is an application programming interface that provides solutions to multi-database programming problem. **Advantages:**
1. It allows s/w developers to develop application without burden of learning MDBS APIs.
2. It has ability to store data for various applications or data from different sources in any database & transparently access or combine the data on an as needed basis.
3. Details of back-end are hidden from users.
- The application interacts with the ODBC driver manager which sends the application calls (SQL statements) to database. The driver manager loads & unloads drivers, performs status checks, & manages multiple connections between applications & data sources.
- ODBC provides a mechanism for creating a virtual DBMS.

**Designing Access Layer Classes:**
- The main idea behind creating an access layer is to create a set of classes that know how to communicate with place(s) where data actually reside. The access layer performs to major tasks:
  1. **Translate the request:** The access layer must be able to translate any data related requests from business layer into appropriate protocol for data access.
  2. **Translate the results:** It must be able to translate the data retrieved back into the appropriate business objects & pass those objects back into business layer.
- **Advantages of Access Layer:** Main advantage is first one while others follows:
  - Design is not tied to any database engine to distributed object technology (CORBA or DCOM).
  - Access layer classes provide easy migration to emerging distributed object technology ("")
  - These classes should be able to address the modest needs of two-tier client–server architectures as well as difficult demands of fine-grained, peer-to-peer distributed object architectures.
- The process of creating an access class for the business classes is identified as follows:
  1. For every business class identified, mirror the business class package – create one access class in access layer package for every respective classes in business layer.
2. **Define relationships** – The same rules as applied among business class objects also applied among access classes

3. **Simplify Classes & relationships** – The main goal is to eliminate redundant or necessary classes or structures. It is done by 2 ways – Eliminate redundant classes & method classes

4. **Iterate and refine**

---

### Layered Approach in DBMS

```
Database, ORB, Internet, etc
Access layer objects

Business layer objects

View layer objects

User
```

### Process of creating Access layer classes

```
O
Mirror business class package
Mirror Super-sub relationships
Normalize classes and relationships
Done
```

### Creating access layer layer classes: storing Persistant attributes

```
Has persistent data
Mirror business Class package
Define relationships
Normalize classes and relationships
Done
```

---

**Another approach** is to let the methods to stored in a program & store only the persistent attributes and modified process constitutes the following steps

1. Determine if the class has any persistent data
2. Mirror the business class package
3. Define relationships
4. Simplify classes and relationships
5. Iterate and refine

---

**Questions: -**

**5 marks**

1. What are the major components of object – oriented design process?
2. Write short notes on object – oriented database.
3. What are the two object oriented design axiom? Explain.
4. List the various rules to avoid bad design and explain.

**10 marks**

1. Explain major corollaries used for designing a system.
2. Discuss the major corollaries used for designing a system in an object oriented method.