ChatGPT

OOP, Database, and Data Structure Interview Questions and Answers

Object-Oriented Programming (OOP)

Basic OOP Questions

- **What is OOP?** OOP is a programming paradigm where software is organized as a collection of *objects* that contain data and methods. In OOP, programs are modeled as interacting real-world entities; each object bundles related data (attributes) and behavior (methods) [1].
- **What is a class and what is an object?** A *class* is a blueprint or template that defines the structure (data) and behavior (methods) of objects. An *object* is an instance of a class with actual values for its properties. In other words, a class is the blueprint, and an object is a concrete instance created from that blueprint [2].
- **What are the main principles of OOP?** The four main principles are **encapsulation**, **abstraction**, **inheritance**, and **polymorphism**. Encapsulation bundles data and methods in one unit and hides internal state [3]; abstraction hides complex details behind a simple interface [4]; inheritance lets a subclass reuse and extend a parent class's code; and polymorphism allows the same operation to behave differently on different classes.
- **What is encapsulation?** Encapsulation is the concept of bundling an object's data (attributes) and the methods that operate on that data into a single unit (class), and restricting direct access to some of an object's components. This "data hiding" ensures that an object's internal state can only be changed in controlled ways (e.g. via getters/setters) [3].
- **What is abstraction?** Abstraction means exposing only essential features of a class while hiding the complex implementation details. It reduces complexity by letting a user interact with a simple interface and ignore underlying code. For example, a class may offer methods without revealing how those methods work internally [4].
- **What is inheritance?** Inheritance is a mechanism where a new class (subclass/derived class) derives from an existing class (superclass/base class). The subclass automatically has the same attributes and methods of its parent and can extend or override them. Inheritance models "is-a" relationships and promotes code reuse [5].
- **What is polymorphism?** Polymorphism ("many forms") means that objects of different classes can be treated through the same interface. In OOP, polymorphism lets a subclass override a parent's method to provide its own behavior, while code can still use the parent type. For example, calling `draw()` on a base `Shape` reference will invoke the correct subclass's `draw()` at runtime [6].
- **What is the difference between method overloading and method overriding?** Overloading occurs when a class has multiple methods with the same name but different parameter lists (signatures) [7]. Overriding occurs when a subclass provides a new implementation of a method that has the same signature as one in its parent class [7]. In short, overloading is compile-time (same class, different params) and overriding is run-time (subclass replaces parent's method).

Intermediate OOP Questions

- **What is an abstract class vs. an interface?** An *abstract class* cannot be instantiated and can include both abstract methods (without body) and concrete methods. It can define common behavior and state for subclasses [8]. An *interface* (in languages like Java) declares a set of methods (all abstract by default, before Java 8) that implementing classes must define [9]. Interfaces cannot hold state (except constants) and support multiple inheritance of type; abstract classes can hold state and allow single inheritance of implementation.
- **What is the "diamond problem" in multiple inheritance?** In C++, if a class inherits from two classes that share a common ancestor, the inheritance diagram forms a diamond shape. This causes ambiguity because the derived class has two copies of the base class's members. The "diamond problem" refers to this ambiguity (e.g. two copies of `fun()` in a class) [10]. C++ resolves it via *virtual inheritance*, which ensures only one shared base instance.
- **What are constructors?** A constructor is a special method (whose name matches the class) that is automatically called when an object is created. Its job is to initialize the new object's attributes to valid states [11]. For example, in C++ a constructor has the same name as the class, and in Python it's named `__init__`. Constructors prepare the object for use by setting up initial values.

Expert OOP Questions

- **What are the SOLID principles?** SOLID is an acronym for five design principles that make software more maintainable:
- **Single-Responsibility Principle** – a class should have only one reason to change (one job).
- **Open-Closed Principle** – software entities should be open for extension but closed for modification.
- **Liskov Substitution Principle** – objects of a superclass should be replaceable by objects of subclasses without affecting correctness.
- **Interface Segregation Principle** – clients should not be forced to depend on interfaces they do not use.
- **Dependency Inversion Principle** – depend on abstractions, not on concrete implementations [12].
- **What is a virtual function?** (C++/similar) A virtual function is a method declared with the `virtual` keyword in a base class, allowing subclasses to override it. It enables dynamic (run-time) polymorphism: when you call a virtual method through a base-class pointer/reference, the subclass's version is invoked at run-time. (Without `virtual`, method calls are resolved by the static type.)
- **Explain runtime vs. compile-time polymorphism.** Compile-time polymorphism is achieved via method overloading or templates; the correct method is chosen by the compiler. Runtime polymorphism is achieved via method overriding (using virtual methods) where the decision of which method to call is made at run-time based on the object's actual class.
- **What is the diamond problem solution?** (C++ specific) Using *virtual inheritance* in C++ ensures that all classes in a diamond hierarchy share a single base class instance. For example: `class Child : virtual public Parent1, virtual public Parent2 { };` ensures there is only one shared `Base` subobject [10].
- **What is the difference between an interface and abstract class?** An abstract class can provide both method declarations and implementations; it may have member variables and constructors [8]. An interface (pre-Java 8) can only declare method signatures (no state), and a class can implement multiple interfaces. (Post-Java 8, interfaces can also have default methods with implementation [9].)

Database

Basic Database Questions

- **What is a database? What is a DBMS?** A *database* is a structured collection of data stored electronically (often in tables) [13] . A *DBMS (Database Management System)* is the software that manages and provides access to the database, handling queries, updates, backups, etc. [13] . Together they allow users to store, retrieve, and update data efficiently.
- **What is a table, row, and column?** In a relational database, data is stored in **tables**. A table has **columns** (fields) with specific data types (e.g. `id`, `name`, `date`) and **rows** (records) which are entries in the table. Each row has one value per column.
- **What is a primary key?** A primary key is one or more columns in a table that uniquely identify each row. Every table should have a primary key so that no two rows have the same primary-key value [14] . For example, `EmployeeID` might be a primary key in an Employees table. The primary key enforces uniqueness and helps link related data across tables.
- **What is a foreign key?** A foreign key is a column (or set of columns) in one table that refers to the primary key of another table [15] . It creates a relationship between the two tables. For example, an `orders` table may have a `customer_id` foreign key that points to the `id` primary key in the `customers` table [15] . This ensures that each order is linked to a valid customer.
- **What is normalization?** Normalization is the process of organizing database tables to reduce redundancy and dependency. It involves splitting tables so that data is stored logically. For example, **1NF** (First Normal Form) requires all column values to be atomic (no repeating groups), **2NF** adds that non-key columns must depend on the whole key, and **3NF** adds that columns must depend only on the primary key (no transitive dependencies) [16] [17] . Normal forms eliminate duplicate data and update anomalies.
- **What is SQL?** SQL (Structured Query Language) is the standard language for querying and modifying relational databases. It includes commands like `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `CREATE TABLE` to manipulate database objects and data.

Intermediate Database Questions

- **What are JOINs?** A JOIN combines rows from two or more tables based on related columns. In SQL:
- **INNER JOIN** returns only rows with matching keys in both tables.
- **LEFT JOIN** returns all rows from the left table and matched rows from the right table (NULL if no match).
- **RIGHT JOIN** returns all rows from the right table and matched rows from the left (NULL if no match).
- **FULL JOIN** returns all rows when there is a match in one of the tables, filling NULLs where there's no match [18] .
- **What is an index?** A database index is a special data structure (often a B-tree) that improves the speed of data retrieval on a table at the cost of extra storage and slower writes [19] . It allows the database to find rows faster without scanning the entire table. For example, an index on the `email` column of a users table lets you search by email quickly. The index stores sorted values of the column with pointers to the rows.

- **What are ACID properties?** ACID (Atomicity, Consistency, Isolation, Durability) are four key properties of database transactions [20].

- *Atomicity* ensures each transaction is "all-or-nothing" (either fully succeeds or fully fails) [21].
- *Consistency* means transactions bring the database from one valid state to another, obeying all rules and constraints [22].
- *Isolation* ensures concurrent transactions do not interfere; each sees a consistent view of the data [23].
- *Durability* means once a transaction is committed, its changes are permanent even if the system crashes.
- **What is a transaction?** A transaction is a sequence of database operations (reads/writes) treated as a single logical unit. Transactions use ACID properties to ensure integrity: if any part fails, the transaction rolls back, leaving the database unchanged.
- **What is the difference between `WHERE` and `HAVING`?** `WHERE` filters rows before any grouping or aggregation; `HAVING` filters groups after an aggregation (it applies conditions to grouped records). For example, `WHERE age > 18` filters individual rows, whereas `HAVING COUNT(*) > 1` would filter groups in a `GROUP BY` query.
- **What is denormalization?** Denormalization is the intentional introduction of redundancy into a database design to improve read performance. For example, duplicating a column from one table into another can avoid a costly JOIN, at the expense of potential data duplication. It's a trade-off used for optimization when necessary.

Expert Database Questions

- **What is the CAP theorem?** In distributed databases, the CAP theorem states that a system can only guarantee two out of three: Consistency, Availability, and Partition tolerance [24]. In the event of a network partition, you must choose between consistency (all nodes see the same data) or availability (system continues to operate) [24]. For example, traditional SQL databases favor consistency and partition tolerance, while many NoSQL systems prefer availability and partition tolerance.

- **Explain 1NF, 2NF, 3NF, BCNF.**

- **1NF:** Each column must be atomic (no repeating or nested data); each row is unique.
- **2NF:** In 1NF and every non-key column depends on the *whole* primary key (no partial dependencies).
- **3NF:** In 2NF and no non-key column depends on another non-key column (no transitive dependencies) [17].

- **BCNF (Boyce-Codd NF):** A stricter form of 3NF where every determinant must be a candidate key [25]. In BCNF, for every functional dependency X→Y, X must be a superkey of the table.

- **What is query optimization?** The database query optimizer determines the most efficient way to execute a SQL query. It analyzes possible query plans (e.g. which indexes to use, join order) to minimize I/O and CPU cost. The chosen "query plan" is then executed. (Typical factors include indexes, statistics, and join strategies.)

- **What is indexing on non-unique columns?** You can create indexes on columns that are not unique to speed up queries. Such an index will group rows by column value and point to all rows with that

value. It speeds up searches but still allows duplicates. Some DBMSes allow *partial indexes* (only index rows meeting a condition) or *function-based indexes* (index on an expression) for specialized optimization.

Data Structures

Basic Data Structures Questions

- **What is an array?** An array is a data structure of fixed-size that stores elements of the same type in contiguous memory locations [26]. You access elements by their index (e.g. `a[0], a[1], ...`). Accessing any element is O(1) time, but inserting or deleting at arbitrary positions takes O(n) because elements may need to be shifted.
- **What is a linked list?** A linked list is a linear data structure where each node contains data and a pointer to the next node [27]. There is no contiguous memory constraint. Linked lists allow efficient insertion and deletion (O(1) if you have a pointer to the node) because you only change pointers [27], but random access is slow (O(n)) since you must traverse from the head.
- **What is a stack?** A stack is a data structure that follows LIFO (last in, first out) order [28]. Elements are added and removed from the *top* only. Common operations: `push` (add top) and `pop` (remove top). Example: function call stacks use this model.
- **What is a queue?** A queue is a data structure that follows FIFO (first in, first out) order [29]. Elements are added at the *rear* and removed from the *front*. Common operations: `enqueue` (add to rear) and `dequeue` (remove from front). Example: print job scheduling uses a queue.
- **What is a tree?** A tree is a hierarchical data structure with nodes connected by edges. A *binary tree* is a tree where each node has at most two children (left and right) [30]. The top node is the root; nodes without children are leaves. Trees are used to represent hierarchical data (e.g. file systems, organization charts).
- **What is a graph?** A graph is a collection of vertices (nodes) and edges connecting pairs of vertices [31]. Graphs can be directed or undirected. They model complex relationships, such as social networks or road maps. Traversals (BFS/DFS) and path algorithms (Dijkstra's) are common graph operations.
- **What is a hash table?** A hash table (hash map) is a data structure for storing key–value pairs with fast access. It uses a *hash function* to map keys to indexes in an array [32]. Ideally, lookups, insertions, and deletions take ~O(1) time. Collisions (different keys hashing to the same index) are handled by methods like chaining or open addressing.

Intermediate Data Structures Questions

- **What is a binary search tree (BST)?** A BST is a binary tree where for each node, all values in the left subtree are less than the node's key, and all values in the right subtree are greater [33]. This ordering enables efficient search, insertion, and deletion (O(log n) on average if balanced). Example: inserting/searching in a BST compares keys and goes left/right accordingly.
- **What is a heap (priority queue)?** A heap is a complete binary tree that satisfies the heap property [34]. In a *min-heap*, every parent's key is $\leq$ its children's keys, so the minimum is at the root; in a *max-heap*, every parent $\geq$ children. Heaps are often implemented as arrays. They support `insert` and `extract-min` (or max) in O(log n) time, making them useful for priority queues.
- **What is a trie?** A trie (prefix tree) is a tree-like data structure for storing a dynamic set of strings (e.g. words) [35]. Each node represents a prefix of some keys. Strings are stored by character at each level,

so common prefixes share paths. Tries allow fast lookup, insertion, and prefix queries (e.g. autocomplete) in time proportional to the string length.

- **What is BFS and DFS (graph traversals)?** Breadth-First Search (BFS) visits graph nodes level by level, starting from a source. It uses a queue: enqueue the start, then repeatedly dequeue a node and enqueue all its unvisited neighbors [36]. Depth-First Search (DFS) explores as far along one branch as possible before backtracking: it can be implemented with recursion or a stack [37]. DFS goes deep first (following one path to the end, then backtracking). Both traverse all reachable nodes, but in different orders.
- **What is collision in hashing?** A collision occurs when two keys hash to the same index in a hash table. Common resolutions are *chaining* (store a linked list of values at that bucket) or *open addressing* (probe to find another free slot). Effective hash functions and good table size are used to minimize collisions.
- **What is a balanced tree?** A balanced tree (like AVL or Red-Black tree) is a binary search tree that automatically keeps its height low ($O(\log n)$). It does so by rotations on insert/delete to ensure the tree remains roughly balanced. This guarantees operations remain $O(\log n)$. (Example: AVL trees rebalance after insertions to maintain height difference ≤ 1.)

Expert Data Structures Questions

- **What is a Disjoint-Set (Union-Find) data structure?** A disjoint-set (union-find) structure maintains a collection of disjoint sets and supports two operations: **find(x)** (identify which set *x* is in) and **union(a, b)** (merge the sets containing *a* and *b*) [38]. It is often implemented with each set as a tree and uses path compression and union by rank to achieve nearly constant amortized time per operation. Disjoint-set is commonly used in Kruskal's MST algorithm and in network connectivity problems [38].
- **What is a self-balancing binary search tree?** Trees like AVL trees or Red-Black trees automatically maintain balance on insert/delete so that the height remains $O(\log n)$. For example, a *Red-Black tree* enforces properties (nodes colored red/black) to ensure no path is more than twice as long as any other, providing balanced performance for search and updates. (No extra citation needed as this is a conceptual answer.)
- **What is a segment tree or Fenwick tree?** These are advanced DS for range queries. A *segment tree* is a binary tree built over an array that allows querying and updating range sums or minimums in $O(\log n)$ by storing intervals at nodes. A *Fenwick tree* (Binary Indexed Tree) is a simpler array-based structure that also supports prefix-sum queries and updates in $O(\log n)$. Both precompute partial sums to answer range queries quickly.
- **What is a skip list?** A skip list is a probabilistic data structure that maintains sorted data and allows fast search/insertion in $O(\log n)$ expected time. It consists of multiple levels of linked lists; each higher level "skips" over nodes in the lower level. By randomly assigning levels to nodes, the structure achieves performance similar to balanced trees, but it is easier to implement. (This can be omitted if too detailed.)

Sources: OOP definitions and concepts from educational references [1] [6] [3] [4] [11] [12]. Database concepts from authoritative articles and tutorials [13] [15] [20] [17] [18] [19] [24]. Data structure definitions from textbooks and tutorials [26] [27] [28] [29] [31] [32] [33] [34] [36] [37] [38].

---

1 30 OOPs Interview Questions and Answers [2025 Updated] - GeeksforGeeks
https://www.geeksforgeeks.org/oops-interview-questions/

2 Classes and Objects in Java - GeeksforGeeks
https://www.geeksforgeeks.org/java/classes-objects-java/

3 Encapsulation in Programming: A Beginner's Guide - Stackify
https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/

4 Abstraction in Programming: A Beginner's Guide - Stackify
https://stackify.com/oop-concept-abstraction/

5 What is Inheritance in Object-Oriented Programming? - Codecademy Blog
https://www.codecademy.com/resources/blog/what-is-inheritance/

6 OOP Concepts for Beginners: What Is Polymorphism - Stackify
https://stackify.com/oop-concept-polymorphism/

7 Overriding vs Overloading in Java | DigitalOcean
https://www.digitalocean.com/community/tutorials/overriding-vs-overloading-in-java

8 9 Difference Between Abstract Class and Interface in Java - GeeksforGeeks
https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/

10 Diamond Problem in C++ - GeeksforGeeks
https://www.geeksforgeeks.org/cpp/diamond-problem-in-cpp/

11 Constructor (object-oriented programming) - Wikipedia
https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)

12 SOLID Design Principles Explained: Building Better Software Architecture | DigitalOcean
https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

13 What is a Database? - Uses, How it Works & Components | Nutanix
https://www.nutanix.com/info/database

14 Primary Key Definition, Explanation & Examples - Explanation & Examples | Secoda
https://www.secoda.co/glossary/primary-key

15 Foreign key - Wikipedia
https://en.wikipedia.org/wiki/Foreign_key

16 17 25 Normal Forms in DBMS - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/normal-forms-in-dbms/

18 SQL Joins (Inner, Left, Right and Full Join) - GeeksforGeeks
https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/

19 Database index - Wikipedia
https://en.wikipedia.org/wiki/Database_index

20 21 22 23 ACID Properties in DBMS - GeeksforGeeks
https://www.geeksforgeeks.org/acid-properties-in-dbms/

24 CAP theorem - Wikipedia
https://en.wikipedia.org/wiki/CAP_theorem

26 Array (data structure) - Wikipedia
https://en.wikipedia.org/wiki/Array_(data_structure)

27 Linked List Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/linked-list-data-structure/

28 Stack Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/stack-data-structure/

29 Queue Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/queue-data-structure/

30 Binary Tree Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/binary-tree-data-structure/

31 Graph Data Structure
https://www.programiz.com/dsa/graph

32 Hash Table Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/hash-table-data-structure/

33 Binary Search Tree - GeeksforGeeks
https://www.geeksforgeeks.org/binary-search-tree-data-structure/

34 Heap: An Efficient Data Structure | by Amir Abdallah | Medium
https://medium.com/@amir21abdallah/heap-an-efficient-data-structure-bad6a36dddd4

35 Trie Data Structure - GeeksforGeeks
https://www.geeksforgeeks.org/dsa/trie-insert-and-search/

36 Breadth First Search or BFS for a Graph - GeeksforGeeks
https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

37 Depth-first search - Wikipedia
https://en.wikipedia.org/wiki/Depth-first_search

38 Disjoint-set data structure - Wikipedia
https://en.wikipedia.org/wiki/Disjoint-set_data_structure