**What is File Storage ?**

**File Storage** refers to a method of storing data in files on a storage medium (like a hard drive, SSD, or cloud storage). Each file contains data and is organized in a hierarchical structure using directories and folders.  
**Examples**: Text files (.txt), spreadsheets (.xlsx), images (.jpg), etc.

* **Access Method**: You access data by opening and reading the file.
* **Use Case**: Simple data storage like logs, documents, media files.

**What is a DBMS (Database Management System)?**

A **DBMS** is software that allows users to define, create, maintain, and control access to databases. It stores data in a structured format (usually tables) and supports complex querying and data manipulation.

* **Examples**: MySQL, Oracle, PostgreSQL, MongoDB.
* **Access Method**: SQL or other query languages.
* **Use Case**: Applications needing structured data, relationships, and fast querying (e.g., banking systems, e-commerce platforms).
* **Key Differences Between File Storage and DBMS**

| **Feature** | **File Storage** | **DBMS** |
| --- | --- | --- |
| **Data Structure** | Unstructured or semi-structured files | Structured tables with relationships |
| **Data Access** | Manual or via file I/O operations | Query-based (e.g., SQL) |
| **Data Integrity** | Hard to enforce | Enforced via constraints and rules |
| **Concurrency** | Limited | Supports multiple users and transactions |
| **Security** | Basic (file permissions) | Advanced (user roles, encryption, etc.) |
| **Scalability** | Limited for large data sets | Designed for scalability and performance |
| **Backup & Recovery** | Manual | Built-in features |
|  |  |  |
|  |  |  |

**DDL - Data Definition Language**

DDL (Data Definition Language) actually consists of SQL commands that can be used for defining, altering and deleting database structures such as tables, indexes and schemas. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [CREATE](https://www.geeksforgeeks.org/sql/sql-create-table/) | Create database or its objects (table, index, function, views, store procedure and triggers) | CREATE TABLE table\_name (column1 data\_type, column2 data\_type, ...); |
| [DROP](https://www.geeksforgeeks.org/sql/sql-drop-truncate/) | Delete objects from the database | DROP TABLE table\_name; |
| [ALTER](https://www.geeksforgeeks.org/sql/sql-alter-add-drop-modify/) | Alter the structure of the database | ALTER TABLE table\_name ADD COLUMN column\_name data\_type; |
| [TRUNCATE](https://www.geeksforgeeks.org/sql/sql-drop-truncate/) | Remove all records from a table, including all spaces allocated for the records are removed | TRUNCATE TABLE table\_name; |
| [COMMENT](https://www.geeksforgeeks.org/sql/sql-comments/) | Add comments to the data dictionary | COMMENT ON TABLE table\_name IS 'comment\_text'; |
| [RENAME](https://www.geeksforgeeks.org/sql/sql-rename-table/) | Rename an object existing in the database | RENAME TABLE old\_table\_name TO new\_table\_name; |

**DQL - Data Query Language**

DQL is used to fetch data from the database. The main command is SELECT, which retrieves records based on the query. The output is returned as a result set (a temporary table) that can be viewed or used in applications.

**DQL Command**

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [SELECT](https://www.geeksforgeeks.org/sql/sql-select-query/) | It is used to retrieve data from the database | SELECT column1, column2, ...FROM table\_name WHERE condition; |
| FROM | Indicates the **table(s)** from which to retrieve data. | SELECT column1 FROM table\_name; |
| [WHERE](https://www.geeksforgeeks.org/sql/sql-where-clause/) | Filters rows **before** any grouping or aggregation | SELECT column1 FROM table\_name WHERE condition; |
| [GROUP BY](https://www.geeksforgeeks.org/sql/sql-group-by/) | Groups rows that have the same values in specified columns. | SELECT column1, AVG\_FUNCTION(column2) FROM table\_name GROUP BY column1; |
| [HAVING](https://www.geeksforgeeks.org/sql/sql-having-clause-with-examples/) | Filters the results of GROUP BY | SELECT column1, AVG\_FUNCTION(column2) FROM table\_name GROUP BY column1 HAVING condition; |
| [DISTINCT](https://www.geeksforgeeks.org/sql/sql-distinct-clause/) | Removes **duplicate rows** from the result set | SELECT DISTINCT column1, column2, ... FROM table\_name; |
| [ORDER BY](https://www.geeksforgeeks.org/sql/sql-order-by/) | Sorts the result set by one or more columns | SELECT column1 FROM table\_name ORDER BY column1 [ASC | DESC]; |
| [LIMIT](https://www.geeksforgeeks.org/mysql/mysql-limit-clause/) | By default, it sorts in **ascending order** unless specified as DESC | SELECT \* FROM table\_name LIMIT number; |

-- select DISTINCT(city) FROM E\_COMMERCE;

--select order\_id,city,amount from E\_COMMERCE

--ORDER BY order\_id DESC;

--select \* from E\_COMMERCE

--group by city;

--select city,customer\_id,SUM(amount) AS TOTAL\_SPENT FROM E\_COMMERCE

--GROUP BY city;

SELECT city,SUM(amount) As REVENUE

FROM E\_COMMERCE

where status = "Delivered"

group by city

having SUM(amount)>2500;  
  
INSERT INTO CUSTOMER (customer\_id, customer\_name, email, phone)

VALUES

(1001, 'Amit Sharma', 'amit.sharma@example.com', '9876543210'),

(1002, 'Priya Mehta', 'priya.mehta@example.com', '9123456780'),

(1003, 'Ravi Kumar', 'ravi.kumar@example.com', '9988776655'),

(1004, 'Sneha Joshi', 'sneha.joshi@example.com', '9090909090'),

(1005, 'Karan Patel', 'karan.patel@example.com', '9191919191'),

(1006, 'Amit Sharma', 'amit.sharma2@example.com', '9876543211'),

(1007, 'Priya Mehta', 'priya.mehta2@example.com', '9123456781'),

(1008, 'Ravi Kumar', 'ravi.kumar2@example.com', '9988776656'),

(1009, 'Sneha Joshi', 'sneha.joshi2@example.com', '9090909091'),

(1010, 'Karan Patel', 'karan.patel2@example.com', '9191919192');

SELECT

E.order\_id,

E.customer\_id,

C.customer\_name,

E.city,

E.status,

E.amount,

E.ORDER\_DATE

FROM

E\_COMMERCE E

INNER JOIN

CUSTOMER C ON E.customer\_id = C.customer\_id;

**Primary Key**

* **Purpose**: Uniquely identifies each record in a table.
* **Uniqueness**: Must be unique for every row.
* **Nulls**: **Cannot** contain NULL values.
* **Count per Table**: Only **one** primary key allowed per table.
* **Indexing**: Automatically creates a **clustered index** (in most databases).

**Unique Key**

* **Purpose**: Ensures that all values in a column (or combination of columns) are unique.
* **Uniqueness**: Must be unique, but not necessarily the main identifier.
* **Nulls**: **Can** contain NULL values (but only one NULL per column in most databases).
* **Count Table**: You can have **multiple** unique keys in a table.
* **Indexing**: Automatically creates a **non-clustered index**.

**UNION:- (Vertical Merge)**

Matching is not required when UNION is used. It Removes Duplicates, Used for merging the similar data vertically.

**FULL OUTER JOIN:- (Horizontal Merge)**

Combines Rows based on Joined Condition, matching is required, It does not remove duplicates.

**CROSS JOIN**

Definition: A CROSS JOIN returns the Cartesian product of two tables. That means every row from the first table is combined with every row from the second table.

Use Case: Useful when you want to pair every item from one set with every item from another set.

**Table A**

| **ID** | **Name** |
| --- | --- |
| **1** | **Alice** |
| **2** | **Bob** |

**Table B**

| **Dept** |
| --- |
| **HR** |
| **IT** |

**Result of CROSS JOIN**

| **ID** | **Name** | **Dept** |
| --- | --- | --- |
| **1** | **Alice** | **HR** |
| **1** | **Alice** | **IT** |
| **2** | **Bob** | **HR** |
| **2** | **Bob** | **IT** |

**FULL OUTER JOIN**

* **Definition**: A FULL OUTER JOIN returns **all rows** from both tables. If there is a match between the tables, it shows the matched data. If not, it fills in NULL for missing matches.
* **Use Case**: Useful when you want to see all data from both tables, including unmatched rows.

**Example:**

**Table A**

| **ID** | **Name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |

**Table B**

| **ID** | **Dept** |
| --- | --- |
| 2 | IT |
| 3 | HR |

**Result of FULL OUTER JOIN on ID**

| **ID** | **Name** | **Dept** |
| --- | --- | --- |
| 1 | Alice | NULL |
| 2 | Bob | IT |
| 3 | NULL | HR |

View is used to store the Virtual Table, Which is called using view name .

CREATE VIEW employee\_salaries AS

SELECT name, salary

from employees;

SELECT \* FROM employee\_salaries;

**What is a Candidate Key?**

A **candidate key** is a **set of one or more columns (attributes)** in a table that can **uniquely identify each row** in that table. It must satisfy two properties:

1. **Uniqueness**: No two rows can have the same value for the candidate key.
2. **Irreducibility (Minimality)**: If you remove any column from the candidate key, it will no longer uniquely identify rows.

**Day 2**

**SQL (Structured Query Language)**

* **Type**: Declarative language
* **Purpose**: Used to query and manipulate data in a database.
* **Functionality**:
  + Performs operations like SELECT, INSERT, UPDATE, DELETE.
  + Defines database structures using CREATE, ALTER, DROP.
* **Execution**: Executes one statement at a time.
* **Used For**: Data manipulation and definition.

**PL/SQL (Procedural Language/SQL)**

* + **Type**: Procedural extension of SQL
  + **Purpose**: Used to write full programs with logic, loops, and conditions.
  + **Functionality**:
    - Supports variables, conditions (IF), loops (FOR, WHILE), and error handling.
    - Can group multiple SQL statements into a single block.
  + **Execution**: Executes a block of code (procedures, functions, triggers).
  + **Used For**: Writing complex business logic and automation.

| **Feature** | **SQL** | **PL/SQL** |
| --- | --- | --- |
| Language Type | Declarative | Procedural |
| Execution | Single statement | Block of statements |
| Control Structures | Not supported | Supported (IF, LOOP, etc.) |
| Error Handling | Not supported | Supported (EXCEPTION block) |
| Use Case | Data operations | Business logic & automation |

Features if PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle's procedural extension to SQL. It combines the data manipulation power of SQL with the processing capabilities of procedural languages. Here are the **key features of PL/SQL**:

**🔑 Core Features of PL/SQL**

1. **Block Structure**
   * PL/SQL code is organized into blocks: DECLARE, BEGIN, EXCEPTION, and END.
   * This modular structure improves readability and maintainability.
2. **Procedural Capabilities**
   * Supports loops (FOR, WHILE, LOOP), conditional statements (IF, CASE), and variables.
   * Enables complex business logic implementation.
3. **Error Handling**
   * Robust exception handling using the EXCEPTION block.
   * Can catch predefined and user-defined exceptions.
4. **Tight Integration with SQL**
   * You can embed SQL statements directly within PL/SQL blocks.
   * Supports DML operations (SELECT, INSERT, UPDATE, DELETE) seamlessly.
5. **High Performance**
   * Reduces network traffic by bundling multiple SQL statements into a single block.
   * Supports bulk operations using FORALL and BULK COLLECT.
6. **Portability**
   * PL/SQL code can run on any Oracle database without modification.
7. **Security**
   * Supports user-defined procedures and functions with controlled access.
   * Can encapsulate business logic securely.
8. **Triggers and Stored Procedures**
   * Allows creation of triggers, procedures, functions, and packages.
   * These can be reused and maintained centrally.
9. **Cursor Management**
   * Supports both implicit and explicit cursors for row-by-row processing.
10. **Packages**
    * Group related procedures, functions, variables, and cursors into a single unit.
    * Promotes modular programming and code reuse.

**2ND September**

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**. It's a set of practices used in modern software development to automate and streamline the process of building, testing, and releasing code.

**🚀 CI (Continuous Integration)**

**Goal:** Automatically integrate code changes from multiple developers into a shared repository several times a day.

**Key Features:**

* Developers push code to a shared repository (e.g., GitHub, GitLab).
* Automated builds and tests run to verify the changes.
* Helps detect bugs early and ensures code quality.

**Benefits:**

* Reduces integration issues.
* Improves collaboration.
* Speeds up development cycles.

**📦 CD (Continuous Delivery / Continuous Deployment)**

There are two types of CD:

**1. Continuous Delivery**

* Code is automatically tested and prepared for release.
* Deployment to production is **manual** but ready at any time.

**2. Continuous Deployment**

* Every change that passes automated tests is **automatically deployed** to production.
* No manual intervention required.

**Benefits:**

* Faster release cycles.
* Reduced human error.
* More reliable and frequent updates.

**🔧 CI/CD Tools**

Some popular tools include:

* **Jenkins**
* **GitHub Actions**
* **GitLab CI/CD**
* **CircleCI**
* **Azure DevOps**
* **Travis CI**

**What is Scrum?**

**Scrum** is a popular **Agile framework** used for developing, delivering, and sustaining complex products. It emphasizes **collaboration**, **flexibility**, and **continuous improvement**. Scrum is especially common in software development but is also used in other industries.

**📐 Scrum Framework Overview**

Scrum is built around **three pillars** and **five key events**, supported by **three roles** and **artifacts**.

**👥 Scrum Roles**

1. **Product Owner**
   * Represents the customer.
   * Manages the **Product Backlog**.
   * Prioritizes features based on business value.
2. **Scrum Master**
   * Facilitates the Scrum process.
   * Removes obstacles.
   * Coaches the team on Agile principles.
3. **Development Team**
   * Cross-functional team that builds the product.
   * Self-organizing and collaborative.

**📅 Scrum Events (Ceremonies)**

1. **Sprint**
   * A time-boxed iteration (usually 2–4 weeks).
   * Goal: deliver a potentially shippable product increment.
2. **Sprint Planning**
   * Team plans what to deliver in the upcoming sprint.
   * Defines the **Sprint Goal** and selects items from the Product Backlog.
3. **Daily Scrum (Stand-up)**
   * 15-minute daily meeting.
   * Team discusses progress, plans, and blockers.
4. **Sprint Review**
   * Held at the end of the sprint.
   * Team demonstrates what was built to stakeholders.
5. **Sprint Retrospective**
   * Team reflects on the sprint.
   * Identifies improvements for the next sprint.

**📦 Scrum Artifacts**

1. **Product Backlog**
   * Ordered list of everything needed in the product.
   * Managed by the Product Owner.
2. **Sprint Backlog**
   * Subset of Product Backlog items selected for the sprint.
   * Includes tasks and Sprint Goal.
3. **Increment**
   * The sum of all completed Product Backlog items during a sprint.
   * Must be in a usable state.

**🧱 Scrum Pillars**

1. **Transparency**
   * Everyone understands the process and progress.
2. **Inspection**
   * Regular checks on progress and quality.
3. **Adaptation**
   * Adjustments made based on feedback and inspection.

**Uses of Scrum**

1. **Software Development**
   * Most common use case.
   * Helps teams deliver working software in short, iterative cycles.
2. **Product Management**
   * Enables continuous delivery of product features.
   * Aligns development with customer needs and feedback.
3. **Project Management**
   * Provides a structured yet flexible approach to managing tasks and timelines.
   * Improves visibility and accountability.
4. **Marketing Campaigns**
   * Used to plan, execute, and evaluate campaigns in short sprints.
   * Encourages creativity and rapid iteration.
5. **Education & Training**
   * Helps organize curriculum development and training programs.
   * Facilitates feedback-driven improvement.
6. **Manufacturing & Engineering**
   * Applied in prototyping and iterative design processes.
   * Enhances collaboration between design and production teams.
7. **Business Operations**
   * Used for process improvement, strategic planning, and team collaboration.
   * Encourages transparency and adaptability.

**🎯 Benefits of Using Scrum**

* **Faster Delivery** of value through short sprints.
* **Improved Collaboration** among cross-functional teams.
* **Greater Flexibility** to adapt to changing requirements.
* **Higher Transparency** through daily stand-ups and sprint reviews.
* **Continuous Improvement** via retrospectives.
* **Customer-Centric Development** with frequent feedback loops.

**Application of Scrum in Project Management**

**1. Iterative Development**

* Projects are broken down into **sprints** (usually 2–4 weeks).
* Each sprint delivers a **working increment** of the product.
* This allows for continuous progress and frequent reassessment.

**2. Clear Roles and Responsibilities**

* **Product Owner** manages the vision and priorities.
* **Scrum Master** ensures the process runs smoothly.
* **Development Team** executes the work.
* This structure improves accountability and communication.

**3. Transparency and Visibility**

* Daily stand-ups, sprint reviews, and retrospectives keep everyone informed.
* Stakeholders can track progress and provide feedback regularly.

**4. Flexibility to Change**

* Scrum embraces change, allowing teams to adapt to new requirements.
* The Product Backlog is continuously refined based on feedback and evolving needs.

**5. Risk Management**

* Frequent inspection and adaptation reduce the risk of project failure.
* Issues are identified early and addressed promptly.

**6. Customer-Centric Approach**

* Regular demos and reviews ensure the product aligns with customer expectations.
* Feedback is incorporated quickly into future sprints.

**7. Continuous Improvement**

* Retrospectives help teams reflect and improve their processes.
* Encourages a culture of learning and growth.

**📌 Example: Scrum in a Software Project**

* **Sprint Planning:** Team selects user stories from the backlog.
* **Daily Scrum:** Team discusses progress and blockers.
* **Sprint Review:** Demo of completed features to stakeholders.
* **Sprint Retrospective:** Team discusses what went well and what to improve.

**The 5 Scrum Events**

**1. Sprint**

* The **core time-boxed cycle** of Scrum (usually 2–4 weeks).
* A fixed period during which a usable product increment is created.
* Provides rhythm and predictability to the team.

**2. Sprint Planning**

* Held at the **start of each sprint**.
* The team decides **what work will be done** and **how** it will be accomplished.
* Defines the **Sprint Goal** and selects items from the **Product Backlog** to move into the **Sprint Backlog**.

**3. Daily Scrum (Daily Stand-up)**

* A **15-minute meeting** held every day of the sprint.
* Team members answer three questions:
  1. What did I do yesterday?
  2. What will I do today?
  3. Are there any blockers?
* Promotes transparency and quick issue resolution.

**4. Sprint Review**

* Held at the **end of the sprint**.
* The team **demonstrates** the completed work to stakeholders.
* Feedback is gathered to improve the product and backlog.

**5. Sprint Retrospective**

* Held **after the Sprint Review** and before the next Sprint Planning.
* The team reflects on the sprint:
  + What went well?
  + What didn’t go well?
  + What can be improved?
* Leads to actionable improvements for the next sprint.

**Kanban** is a visual workflow management method used to optimize and improve processes. Originating from **Toyota's manufacturing system**, it is now widely used in **Agile project management**, especially in software development, IT operations, and service delivery.

Kanban focuses on **visualizing work**, **limiting work in progress (WIP)**, and **maximizing flow** — helping teams deliver value more efficiently.

**📊 Core Elements of Kanban**

1. **Kanban Board**
   * A visual tool that shows work items as cards.
   * Columns represent stages of the workflow (e.g., To Do, In Progress, Done).
2. **Work Items**
   * Tasks or user stories represented as cards on the board.
3. **WIP Limits**
   * Restricts the number of tasks in each column to avoid overload.
4. **Flow Management**
   * Focuses on smooth movement of tasks from start to finish.
5. **Continuous Delivery**
   * Unlike Scrum, Kanban doesn’t use time-boxed sprints — work is delivered continuously.

**✅ Advantages of Kanban**

1. **Visual Clarity**
   * Everyone can see the status of tasks at a glance.
2. **Flexibility**
   * No fixed iterations; tasks can be added or reprioritized anytime.
3. **Improved Efficiency**
   * WIP limits prevent bottlenecks and reduce multitasking.
4. **Faster Delivery**
   * Continuous flow allows quicker turnaround of tasks.
5. **Better Collaboration**
   * Teams can easily coordinate and communicate using the board.
6. **Scalability**
   * Can be applied to individual teams or across departments.
7. **Data-Driven Improvements**
   * Metrics like lead time and cycle time help optimize performance.

DAY 4

**Generative AI (Gen AI)** refers to a class of artificial intelligence models designed to create new content—such as text, images, audio, video, or code—based on patterns learned from existing data. These models don't just analyze or classify data; they *generate* new data that resembles the training input.

Popular examples include:

* **ChatGPT** (text generation)
* **DALL·E** (image generation)
* **GitHub Copilot** (code generation)
* **Sora by OpenAI** (video generation)

**✅ Advantages of Gen AI**

1. **Content Creation at Scale**  
   Speeds up writing, designing, coding, and other creative tasks.
2. **Personalization**  
   Can tailor content (e.g., marketing emails, product recommendations) to individual users.
3. **Automation of Repetitive Tasks**  
   Helps automate customer support, report generation, and data summarization.
4. **Rapid Prototyping**  
   Useful in software development, design, and product ideation.
5. **Accessibility**  
   Assists people with disabilities (e.g., generating alt text for images, converting speech to text).
6. **Cost Efficiency**  
   Reduces the need for large teams for certain tasks, saving time and money.

DAY 5

**1. JDK (Java Development Kit)**

* **What it is**: A software development kit used to develop Java applications.
* **Includes**:
  + **JRE (Java Runtime Environment)**
  + **Development tools** like javac (Java compiler), javadoc, jar, and debugger tools.
* **Use case**: Needed by developers to write, compile, and debug Java programs.

**2. JRE (Java Runtime Environment)**

* **What it is**: A package that provides the environment to run Java applications.
* **Includes**:
  + **JVM (Java Virtual Machine)**
  + **Core libraries** and other files needed to run Java programs.
* **Use case**: Needed by users who want to run Java applications but not develop them.

**3. JVM (Java Virtual Machine)**

* **What it is**: A virtual machine that runs Java bytecode.
* **Key role**: Converts bytecode into machine code for the host system.
* **Platform-independent**: Enables Java’s “write once, run anywhere” capability.

import java.util.Scanner;

public class MathOperations {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Input two numbers

        System.out.print("Enter first number: ");

        double num1 = scanner.nextDouble();

        System.out.print("Enter second number: ");

        double num2 = scanner.nextDouble();

        // Display operation choices

        System.out.println("Choose an operation:");

        System.out.println("1 - Addition (+)");

        System.out.println("2 - Subtraction (-)");

        System.out.println("3 - Multiplication (\*)");

        System.out.println("4 - Division (/)");

        int choice = scanner.nextInt();

        double result;

        // Perform selected operation

        switch (choice) {

            case 1:

                result = num1 + num2;

                System.out.println("Result: " + result);

                break;

            case 2:

                result = num1 - num2;

                System.out.println("Result: " + result);

                break;

            case 3:

                result = num1 \* num2;

                System.out.println("Result: " + result);

                break;

            case 4:

                if (num2 != 0) {

                    result = num1 / num2;

                    System.out.println("Result: " + result);

                } else {

                    System.out.println("Error: Division by zero is not allowed.");

                }

                break;

            default:

                System.out.println("Invalid choice.");

        }

        scanner.close();

    }

}

In Java, **data types** are divided into two main categories:

**1. Primitive Data Types**

These are the most basic data types built into the language.

| **Data Type** | **Size** | **Description** | **Example** |
| --- | --- | --- | --- |
| byte | 1 byte | Stores whole numbers from -128 to 127 | byte b = 100; |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 | short s = 1000; |
| int | 4 bytes | Stores whole numbers from -2³¹ to 2³¹-1 | int i = 50000; |
| long | 8 bytes | Stores whole numbers from -2⁶³ to 2⁶³-1 | long l = 100000L; |
| float | 4 bytes | Stores fractional numbers, up to 7 digits | float f = 5.75f; |
| double | 8 bytes | Stores fractional numbers, up to 15 digits | double d = 19.99; |
| char | 2 bytes | Stores a single character or Unicode | char c = 'A'; |
| boolean | 1 bit | Stores true or false values | boolean flag = true; |

**2. Non-Primitive (Reference) Data Types**

These are more complex and include classes, arrays, interfaces, etc.

* **String**: Represents a sequence of characters.
* **Array**: A collection of elements of the same type.
* **Class**: A blueprint for creating objects.
* **Interface**: A reference type used to achieve abstraction and multiple inheritance.

**Bonus Tip:**

You can use the var keyword (Java 10+) for type inference:

import java.util.Scanner;

public class NumberInput {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int number;

do {

System.out.print("Please enter a number: ");

while (!scanner.hasNextInt()) {

System.out.println("That's not a valid number. Try again.");

System.out.print("Please enter a number: ");

scanner.next(); // discard invalid input

}

number = scanner.nextInt();

} while (false); // loop runs only once

System.out.println("You entered: " + number);

scanner.close();

}

}

**SWITCH CASE EXAMPLE**

import java.util.Scanner;

public class DayOfWeek {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number (1 to 7): ");

int day = scanner.nextInt();

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

case 4:

System.out.println("Thursday");

break;

case 5:

System.out.println("Friday");

break;

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;

default:

System.out.println("Invalid input! Please enter a number between 1 and 7.");

}

scanner.close();

}

}  
  
// Define a class called Car

class Car {

// Fields (properties)

String brand;

int year;

// Constructor to initialize the object

Car(String brand, int year) {

this.brand = brand;

this.year = year;

}

// Method to display car details

void displayInfo() {

System.out.println("Brand: " + brand);

System.out.println("Year: " + year);

}

}

// Main class to create objects

public class Main {

public static void main(String[] args) {

// Create an object of Car class

Car myCar = new Car("Toyota", 2022);

// Call method using the object

myCar.displayInfo();

}

}

EXCEPTION CODES

import java.io.\*;

import java.util.\*;

public class ExceptionDemo {

    public static void main(String[] args) {

        // 1. ArithmeticException

        try {

            int result = 10 / 0;

        } catch (ArithmeticException e) {

            System.out.println("ArithmeticException: " + e.getMessage());

        }

        // 2. ArrayIndexOutOfBoundsException

        try {

            int[] arr = new int[5];

            int value = arr[10];

        } catch (ArrayIndexOutOfBoundsException e) {

            System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());

        }

        // 3. NullPointerException

        try {

            String str = null;

            str.length();

        } catch (NullPointerException e) {

            System.out.println("NullPointerException: " + e.getMessage());

        }

        // 4. NumberFormatException

        try {

            int num = Integer.parseInt("abc");

        } catch (NumberFormatException e) {

            System.out.println("NumberFormatException: " + e.getMessage());

        }

        // 5. ClassCastException

        try {

            Object obj = new Integer(10);

            String s = (String) obj;

        } catch (ClassCastException e) {

            System.out.println("ClassCastException: " + e.getMessage());

        }

        // 6. FileNotFoundException (Checked)

        try {

            FileReader fr = new FileReader("nonexistent.txt");

        } catch (FileNotFoundException e) {

            System.out.println("FileNotFoundException: " + e.getMessage());

        }

        // 7. IOException (Checked)

        try {

            File tempFile = File.createTempFile("temp", ".txt");

            FileWriter fw = new FileWriter(tempFile);

            fw.write("Hello");

            fw.close();

        } catch (IOException e) {

            System.out.println("IOException: " + e.getMessage());

        }

        // 8. InterruptedException (Checked)

        try {

            Thread.sleep(100);

        } catch (InterruptedException e) {

            System.out.println("InterruptedException: " + e.getMessage());

        }

        // 9. IllegalArgumentException

        try {

            throw new IllegalArgumentException("Invalid argument passed");

        } catch (IllegalArgumentException e) {

            System.out.println("IllegalArgumentException: " + e.getMessage());

        }

        // 10. IllegalStateException

        try {

            throw new IllegalStateException("Illegal state encountered");

        } catch (IllegalStateException e) {

            System.out.println("IllegalStateException: " + e.getMessage());

        }

        // 11. IndexOutOfBoundsException

        try {

            List list = new ArrayList<>();

            list.get(1);

        } catch (IndexOutOfBoundsException e) {

            System.out.println("IndexOutOfBoundsException: " + e.getMessage());

        }

        // 12. NoSuchElementException

        try {

            Scanner scanner = new Scanner("");

            scanner.nextLine();

        } catch (NoSuchElementException e) {

            System.out.println("NoSuchElementException: " + e.getMessage());

        }

        // 13. SecurityException

        try {

            System.getSecurityManager().checkExit(0);

        } catch (SecurityException e) {

            System.out.println("SecurityException: " + e.getMessage());

        }

        // 14. UnsupportedOperationException

        try {

            List fixedList = Arrays.asList("A", "B");

            fixedList.add("C");

        } catch (UnsupportedOperationException e) {

            System.out.println("UnsupportedOperationException: " + e.getMessage());

        }

        // 15. StackOverflowError

        try {

            recursiveCall();

        } catch (StackOverflowError e) {

            System.out.println("StackOverflowError: " + e.getMessage());

        }

        // 16. OutOfMemoryError

        try {

            List memoryHog = new ArrayList<>();

            while (true) {

                memoryHog.add(new int[1000000]);

            }

        } catch (OutOfMemoryError e) {

            System.out.println("OutOfMemoryError: " + e.getMessage());

        }

    }

    public static void recursiveCall() {

        recursiveCall(); // causes StackOverflowError

    }

}

CHECKED AND UNCHECKED EXCEPTION

try {

FileReader file = new FileReader("data.txt");

} catch (FileNotFoundException e) {

System.out.println("File not found!");

}

Checked Exceptions

These are **checked at compile time**, meaning the compiler forces you to handle them.

* **Must be caught or declared** using try-catch or throws
* Typically represent **recoverable conditions** (e.g., file not found, network issues)
* Subclass of Exception but **not** RuntimeException
* **Examples**:
* IOException
* SQLException
* FileNotFoundException

try { FileReader file = new FileReader("data.txt"); } catch (FileNotFoundException e) { System.out.println("File not found!"); }

Unchecked Exceptions

These occur **at runtime**, and the compiler doesn’t require you to handle them.

* **No need to catch or declare**
* Usually caused by **programming errors** (e.g., logic flaws, null references)
* Subclass of RuntimeException
* **Examples**:
* NullPointerException
* ArithmeticException
* ArrayIndexOutOfBoundsException

int[] arr = new int[3]; System.out.println(arr[5]); // Throws ArrayIndexOutOfBoundsException

CUSTOM EXCEPTION  
// Step 1: Define the custom exception

class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

// Step 2: Use the custom exception in a class

class BankAccount {

private double balance;

public BankAccount(double balance) {

this.balance = balance;

}

public void withdraw(double amount) throws InsufficientFundsException {

if (amount > balance) {

throw new InsufficientFundsException(

"Withdrawal of ₹" + amount + " denied. Available balance: ₹" + balance

);

}

balance -= amount;

System.out.println("Withdrawal successful. Remaining balance: ₹" + balance);

}

}

// Step 3: Test the custom exception

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount(5000);

try {

account.withdraw(7000); // This will trigger the exception

} catch (InsufficientFundsException e) {

System.out.println("Error: " + e.getMessage());

}

}

}  
An array is a fixed-size, homogeneous data structure that stores a collection of similar elements in contiguous memory locations, allowing for quick access to elements using an index number. Its limitations include fixed size (requiring memory to be potentially wasted or insufficient), homogeneity (only storing one data type), and inefficient element insertion/deletion, which requires shifting other elements.   
  
import java.util.Scanner;

public class ShoppingList {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Define the maximum number of items

        final int MAX\_ITEMS = 10;

        String[] shoppingList = new String[MAX\_ITEMS];

        int itemCount = 0;

        while (true) {

            System.out.println("\nShopping List Menu:");

            System.out.println("1. Add Item");

            System.out.println("2. View List");

            System.out.println("3. Remove Item");

            System.out.println("4. Exit");

            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();

            scanner.nextLine(); // consume newline

            switch (choice) {

                case 1:

                    if (itemCount < MAX\_ITEMS) {

                        System.out.print("Enter item to add: ");

                        String item = scanner.nextLine();

                        shoppingList[itemCount] = item;

                        itemCount++;

                        System.out.println("Item added.");

                    } else {

                        System.out.println("List is full!");

                    }

                    break;

                case 2:

                    System.out.println("\nYour Shopping List:");

                    for (int i = 0; i < itemCount; i++) {

                        System.out.println((i + 1) + ". " + shoppingList[i]);

                    }

                    break;

                case 3:

                    System.out.print("Enter item number to remove: ");

                    int removeIndex = scanner.nextInt() - 1;

                    if (removeIndex >= 0 && removeIndex < itemCount) {

                        for (int i = removeIndex; i < itemCount - 1; i++) {

USING STACK  
import java.util.Scanner;

public class ShoppingList {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Define the maximum number of items

        final int MAX\_ITEMS = 10;

        String[] shoppingList = new String[MAX\_ITEMS];

        int itemCount = 0;

        while (true) {

            System.out.println("\nShopping List Menu:");

            System.out.println("1. Add Item");

            System.out.println("2. View List");

            System.out.println("3. Remove Item");

            System.out.println("4. Exit");

            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();

            scanner.nextLine(); // consume newline

            switch (choice) {

                case 1:

                    if (itemCount < MAX\_ITEMS) {

                        System.out.print("Enter item to add: ");

                        String item = scanner.nextLine();

                        shoppingList[itemCount] = item;

                        itemCount++;

                        System.out.println("Item added.");

                    } else {

                        System.out.println("List is full!");

                    }

                    break;

                case 2:

                    System.out.println("\nYour Shopping List:");

                    for (int i = 0; i < itemCount; i++) {

                        System.out.println((i + 1) + ". " + shoppingList[i]);

                    }

                    break;

                case 3:

                    System.out.print("Enter item number to remove: ");

                    int removeIndex = scanner.nextInt() - 1;

                    if (removeIndex >= 0 && removeIndex < itemCount) {

                        for (int i = removeIndex; i < itemCount - 1; i++) {

ArrayList :

String[] shoppingArray = new String[5];

shoppingArray[0] = "Milk";

shoppingArray[1] = "Bread";  
   
VECTOR:

import java.util.Vector;

Vector shoppingVector = new Vector<>();

shoppingVector.add("Milk");

shoppingVector.add("Bread");

**EXCEPTIONAL HANDLING IN PYTHON**

try:

num =int(input("Enter a Number: "))

print(10/num)

except ZeroDivisionError:

print("Cannot divide by Zero. ")

except ValueError:

print("Invalid Input ! Please Enter a Number. ")

else:

print("Division Successful.")

finally:

print("Excution Complete")

ZeroDivisionError :Raised when dividing by Zero

IndexError :Raised when accessing an invalid list Index

KeyError:Raised when a Dictionary key is not Found.

TypeError:Raised when an operation is applied to an inappropriate type

ValueError:Raised when a Fuction Receives an Argument of Correct type but inappropriate value.  
  
FILE HANDLING

File handling refers to reading from, writing to, and managing files stored on disk. Python provides a simple yet powerful interface for this using built-in functions and modules.  
file = open("example.txt", "mode")

WITH OPEN(“data.txt”,”r”) as f:

Content = f.read();

What is TDD ?

TDD is the powerful software development technique that flips the traditional coding process on its head. Instead of writing code first and testing later, TDD starts with the test cases before the actual code.  
**Test-Driven Development (TDD)** is a methodology where you:

1. **Write a test** for a specific functionality.
2. **Run the test** — it should fail initially (since the code doesn’t exist yet).
3. **Write the minimum code** needed to make the test pass.
4. **Refactor the code** for optimization without changing its behavior.
5. **Repeat** the cycle for each new feature or functionality.

This cycle is often called **Red-Green-Refactor**: | Phase | De

Why Use TDD?

* Ensures **high test coverage** — every piece of code is tested.
* Encourages **modular, clean design**.
* Helps catch bugs early.
* Builds **confidence** in code changes and refactoring.

🆚 TDD vs Traditional Testing

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

TDD is especially useful in **agile environments**, and pairs beautifully with **unit testing frameworks** like JUnit (Java), PyTest (Python), or NUnit (.NET).

Would you like to see a simple Java or Python example of TDD in action?

Testing—especially in software development—is the process of evaluating a system or its components to determine whether it meets specified requirements and functions correctly. Think of it as a quality assurance checkpoint that ensures your code or application behaves as expected before it reaches users.  
  
Web services are **software components** that allow applications to communicate with each other over the internet using **standardized protocols** like HTTP, XML, and SOAP. They enable **machine-to-machine interaction**, regardless of the programming language or platform.

**In simple terms:**

A web service is like a waiter in a restaurant—it takes your request (order), delivers it to the kitchen (server), and brings back the response (your food).

🧩 Key Components of Web Services

* **SOAP (Simple Object Access Protocol):** XML-based protocol for exchanging structured information.
* **WSDL (Web Services Description Language):** Describes the service and how to access it.
* **UDDI (Universal Description, Discovery, and Integration):** Directory for discovering web services.

✅ Advantages of Web Services

Advantages of Web Services

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

WSDL (Web Services Description Language)

**What it is:**  
An **XML-based language** used to describe the functionality offered by a web service.

**Purpose:**

* Specifies **what** the service does
* Describes **how** to call it (methods, parameters, return types)
* Indicates **where** it's located (URL endpoint)

**Key Elements:**

* types: Data types used in the service
* message: Input/output messages
* portType: Operations (like functions)
* binding: Protocols used (e.g., SOAP over HTTP)
* service: Actual endpoint URL

UDDI (Universal Description, Discovery, and Integration)

**What it is:**  
A **directory service** where businesses can **publish and discover** web services.

**Purpose:**

* Acts like a **yellow pages** for web services
* Helps clients find services based on business, category, or technical specs
* Stores metadata and references to WSDL files

**Structure:**

* **White Pages**: Business contact info
* **Yellow Pages**: Industry classification
* **Green Pages**: Technical details (WSDL, binding info)

**How WSDL and UDDI Work Together:**

* WSDL defines the **interface** of the service
* UDDI registers the **location and metadata** of that service
* Developers or systems can query UDDI to find services and retrieve their WSDL for integration

API

Advantages of APIs

* **Cross-platform communication**: APIs allow systems built in different languages or platforms to interact seamlessly.
* **Faster development**: Developers can plug into existing services (like payment gateways or stock feeds) instead of building from scratch.
* **Modular architecture**: APIs promote separation of concerns, making systems easier to maintain and scale.
* **Real-time data access**: APIs enable live updates, which is critical for financial markets and commodity tracking.
* **Security control**: APIs expose only necessary endpoints, helping protect sensitive backend logic.
* **Automation-friendly**: APIs can be used to automate workflows, data syncs, and integrations across services.
* **Ecosystem expansion**: Businesses can expose APIs to partners or third parties, creating new revenue streams or integrations.
* **Microservices support**: APIs are the backbone of microservices, allowing independent deployment and scaling of components.

Disadvantages of APIs

* **Security vulnerabilities**: If not properly secured, APIs can be exploited for unauthorized access or data leaks.
* **Versioning headaches**: Maintaining backward compatibility across API versions can be complex and error-prone.
* **Rate limits and throttling**: Many APIs restrict how often they can be called, which can affect performance under load.
* **Dependency risks**: Relying on third-party APIs means your system is vulnerable to their downtime or changes.
* **Latency issues**: Network delays or inefficient endpoints can slow down response times, especially in real-time systems.
* **Complex documentation**: Poorly written or overly technical API docs can make integration difficult for developers.
* **Over-fetching or under-fetching**: REST APIs sometimes return too much or too little data, requiring extra calls or filtering.

If you're building or analyzing systems that rely on external data—like commodity prices or financial dashboards—APIs are essential tools. Want to explore REST vs GraphQL next, or dive into how APIs are used in algo trading platforms? I’m ready when you are.

GET   
GET Method

* Used to **retrieve data** from a server.
* It’s **read-only**—doesn’t modify anything on the server.
* Parameters are sent in the **URL**, often as query strings.
* Responses are usually **cached** and can be bookmarked.
* Ideal for fetching things like stock prices, commodity data, or user profiles.
* Should **never be used** for sensitive data (e.g., passwords) since it's visible in the URL.

POST Method

* Used to **send data to the server** to create a new resource.
* Data is sent in the **body** of the request, not the URL.
* Not cached, not bookmarked, and more secure than GET.
* Commonly used for submitting forms, creating new users, or placing orders.
* **Not idempotent**—calling it multiple times may create multiple resources.  
    
    **100 Continue**: The server received the request headers and the client should proceed.
* **101 Switching Protocols**: The server is switching to a different protocol as requested.
* **103 Early Hints**: Used to preload resources before the final response is ready.
* 🟢 2xx – Success
* **200 OK**: The request was successful.
* **201 Created**: A new resource was successfully created (typically after a POST).
* **202 Accepted**: The request was accepted but not yet processed.
* **204 No Content**: The request was successful, but there’s no content to return.

xx – Redirection

* **301 Moved Permanently**: The resource has been permanently moved to a new URL.
* **302 Found**: Temporary redirect to another URL.
* **304 Not Modified**: The resource hasn’t changed since the last request.

🔴 4xx – Client Errors

* **400 Bad Request**: The server couldn’t understand the request due to invalid syntax.
* **401 Unauthorized**: Authentication is required and has failed or not been provided.
* **403 Forbidden**: The server understood the request but refuses to authorize it.
* **404 Not Found**: The requested resource doesn’t exist.
* **405 Method Not Allowed**: The HTTP method is not supported for the requested resource.
* **408 Request Timeout**: The server timed out waiting for the request.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* TESTNG  
  Test cases & Test Suites

Grouping of test Cases

Prioritize

Parameterizations

Parallel Testing

Reports

1)TestNG execute test methods based on alphabetical order

2)@Test (priority = num) controls the order of execution

3) once you provide priority to the test methods, then order of methods is not considered

4) priorities can be random numbers (no need to have a consecutive numbers)

If any priority is not provided, then it will execute first as the default priority is zero.

5)if two methods having same priority then it will again consider the alphabetical order.

Selenium WebDriver Architecture (Selenium 3 & 4)

🔧 Core Components

* **Selenium Client Libraries**
  + Language-specific bindings for Java, Python, C#, Ruby, etc.
  + Allow writing test scripts in preferred programming language.
* **JSON Wire Protocol (Selenium 3)** / **W3C WebDriver Protocol (Selenium 4)**
  + Standardized communication between client libraries and browser drivers.
  + JSON format used to send commands over HTTP (Selenium 3).
  + Selenium 4 adopts W3C protocol for better browser compatibility.
* **Browser Drivers**
  + Act as a bridge between Selenium commands and the actual browser.
  + Examples: ChromeDriver, GeckoDriver (Firefox), EdgeDriver.
  + Drivers translate JSON commands into native browser actions.
* **Real Browsers**

 Where actual automation happens (Chrome, Firefox, Safari, Edge).

* WebDriver interacts with browser UI elements like buttons, forms, links.

🚀 Execution Flow

1. Test script written using Selenium Client Library.
2. Commands converted to JSON and sent via HTTP to the browser driver.
3. Browser driver receives commands and interacts with the browser.
4. Browser performs actions and sends response back to the driver.
5. Driver relays response to the client library, completing the loop.

🌟 Enhancements in Selenium 4

* Native support for **W3C WebDriver standard**.
* Improved **Grid architecture** for parallel test execution.
* Enhanced **DevTools Protocol** integration for Chrome debugging.

**mplicit Wait**

* Applies globally to all elements in the WebDriver instance.
* Tells WebDriver to wait for a certain amount of time before throwing a NoSuchElementException.
* Once set, it remains active for the entire session unless changed.
* Does **not** wait for any specific condition—just delays until the element appears.
* Syntax:

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

Explicit Wait  
  
 Applies to specific elements or conditions.

* Waits until a particular condition is met (e.g., element is clickable, visible, etc.).
* Uses WebDriverWait in combination with ExpectedConditions.
* More precise and customizable than implicit wait.
* Syntax:

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10)); wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));

1)agile product owner, plsql procedure funtions, triggers,scrum master, agile methodology,implicit curson and explicit cursor,oops, data types, python questionaire