

Chapter 1: Getting Familiar with Software Development

Learning Outcomes:

- Understand Software Development, types, life cycle, and development models
- Learn web-based software application and its concepts
- Understand the Building Blocks of Frontend web application Development: HTML, CSS, & JS
- Create a responsive static website

1.1 Introduction to Software Development

1.1.1 What is Software Development

Software development refers to a set of computer science activities dedicated to the process of creating, designing, deploying, and supporting software. Software itself is the set of instructions or programs that tell a computer what to do. It is independent of hardware and makes computers programmable.

Basically, there are 3 types of Software:

- **System Software:** It sits between the hardware and the application software. It provides core functions such as operating systems, disk management, utilities, hardware management, and other operational necessities. Operating systems like Windows, macOS, Android, and iOS are examples of system software.
- **Programming Software:** It gives programmers tools such as text editors, compilers, linkers, debuggers, and other tools to create code. It performs specific tasks to keep the computer running. Utility software is always running in the background. Examples of utility software are security and optimization programs.
- **Application Software:** This is everything else! Anything that is not an operating system or a utility is an application or app. So a word processor, spreadsheet, web browser, and graphics software are all examples of application software, and they can do many specific tasks.
There are different types of application software such as Desktop Based Software Application and Web Based Software Application. The desktop Based application will be installed on the system and then it can be accessed

and utilized on that system only while the Web Based application can be used from anywhere using the internet services.

Here, In our course, we are going to develop the Web-based Software Application.

1.1.2 Software Development Life Cycle

Software Development Life Cycle (SDLC) is a process the software industry uses to design, develop and test high-quality software. The SDLC aims to produce high-quality software that meets or exceeds customer expectations and reaches completion within times and cost estimates. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing developing, maintaining, replacing, and altering or enhancing specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

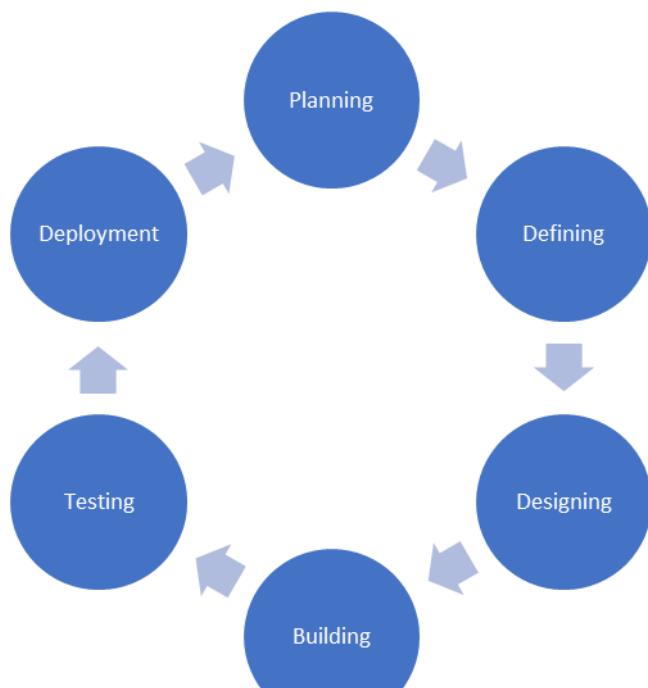


Image: Software Development Life Cycle

➤ Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys, and domain experts in the industry. This information is then used to plan the basic project approach and to conduct a product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of

the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

➤ **Stage 2: Defining Requirements**

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved by the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

➤ **Stage 3: Designing the Product Architecture**

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually, more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters such as risk assessment, product robustness, design modularity, budget, and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third-party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of details in DDS.

➤ **Stage 4: Building or Developing the Product**

In this stage of SDLC, the actual development starts, and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high-level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

➤ **Stage 5: Testing the Product**

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing-only stage of the product where product defects are reported, tracked, fixed, and retested until the product reaches the quality standards defined in the SRS.

➤ **Stage 6: Deployment in the Market and Maintenance**

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

1.1.3 Software Development Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred to as "Software Development Process Models". Each process model follows a series of steps unique to its type to ensure success in the process of software development. Following are the most important and popular SDLC models followed in the industry: Waterfall Model, Iterative Model, Prototype Model, Spiral Model, and Agile Model. We will learn each model in detail.

Waterfall Model:

The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step) and that the output of the stage is consistent with the overall requirements of the system.

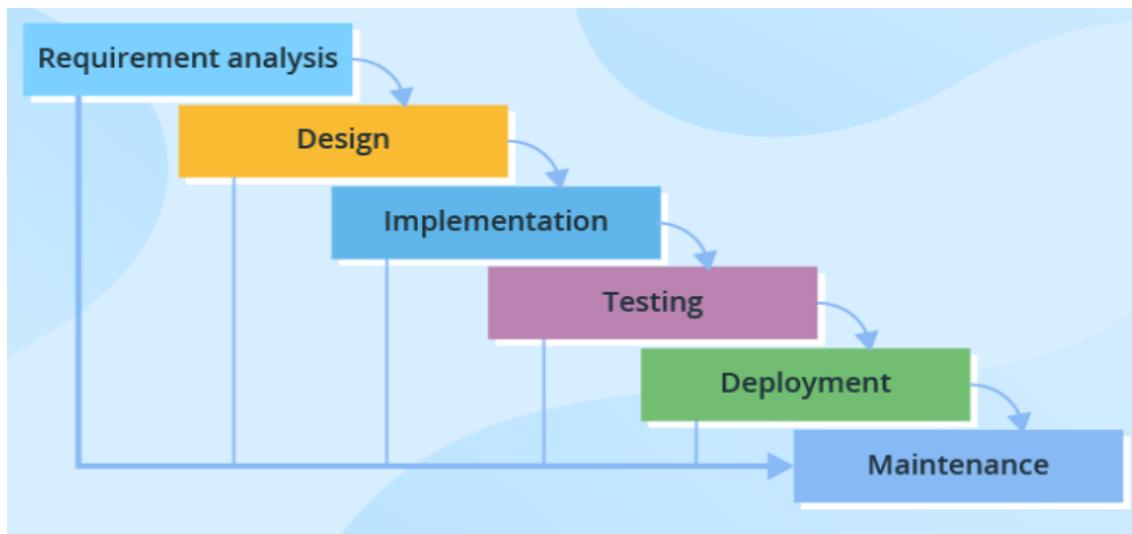


Image: Waterfall Model

Reference: <https://www.javatpoint.com/software-engineering-software-development-life-cycle>

Advantages:

- Before the next phase of development, each phase must be completed
- Suited for smaller projects where requirements are well defined

Disadvantages:

- An error can be fixed only during the phase
- The testing period comes quite late in the developmental process

Iterative Model:

It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.

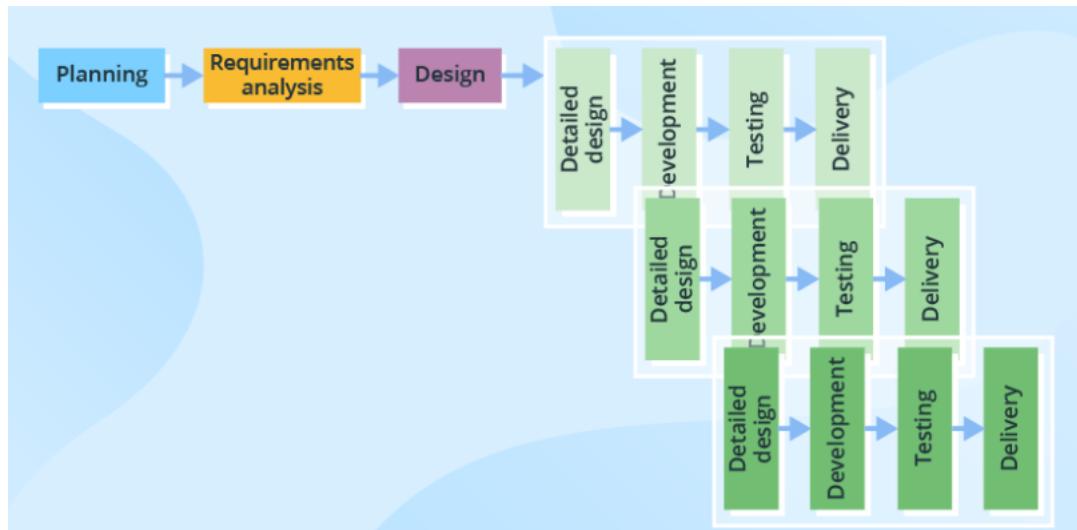


Image: Iterative Model

Reference: <https://www.javatpoint.com/software-engineering-software-development-life-cycle>

Advantages:

- Some working functionality can be developed and early in the software development life cycle (SDLC).
- It is easily adaptable to the ever changing needs of the project as well as the client.
- It is best suited for agile organisations.
- It is more cost effective to change the scope or requirements in Iterative model.
- Parallel development can be planned.

Disadvantages:

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- It is not suitable for smaller projects.
- Highly skilled resources are required for skill analysis.

Prototype Model:

The prototyping model starts with requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.

A 'quick design' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.

Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continues until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

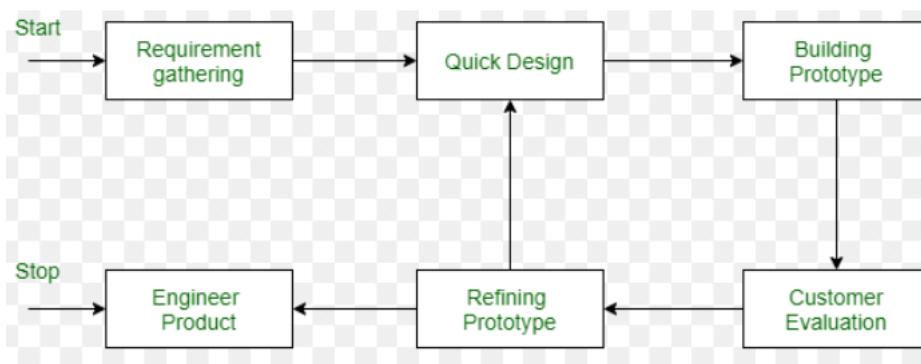


Image: Prototype Model

Reference: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-prototype-model/>

Advantages:

- This model is flexible in design.
- It is easy to detect errors.
- We can find missing functionality easily.
- There is scope for refinement, which means new requirements can be easily accommodated.
- It can be reused by the developer for more complicated projects in the future.

Disadvantages:

- This model is costly.
- It has poor documentation because of continuously changing customer requirements.
- There may be too much variation in requirements.
- Customers sometimes demand the actual product to be delivered soon after seeing an early prototype.

Spiral Model:

The spiral model is a risk-driven process model. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, in addition, the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant).

The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

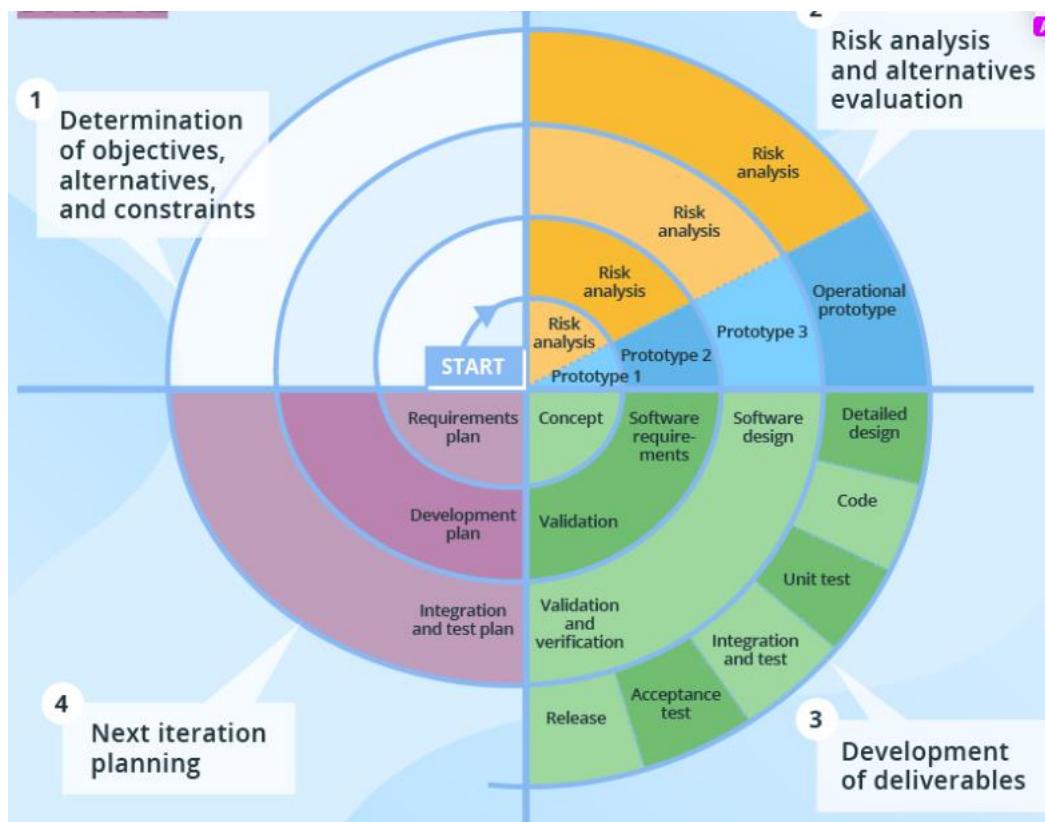


Image: Spiral Model

Reference: <https://www.javatpoint.com/software-engineering-software-development-life-cycle>

Advantages:

- Software is produced early in the software life cycle.
- Risk handling is one of important advantages of the Spiral model, it is best development model to follow due to the risk analysis and risk handling at every phase.
- Flexibility in requirements. In this model, we can easily change requirements at later phases and can be incorporated accurately. Also, additional Functionality can be added at a later date.

Disadvantages:

- It is not suitable for small projects as it is expensive.
- It is much more complex than other SDLC models. The process is complex.

- Too much dependable on Risk Analysis and requires highly specific expertise.

Difficulty in time management. As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Agile Model:

Agile methodology is a practice that promotes continuous interaction of development and testing during the SDLC process of any project. The Agile method divides the entire project into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:

It is difficult to think in advance about which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.

For many types of software, design, and development are interleaved. Both activities should be performed in tandem so that design models are proven as they are created. It is difficult to think about how much design is necessary before construction is used to test the configuration.

Analysis, design, development, and testing are not as predictable (from a planning point of view) as we might like.

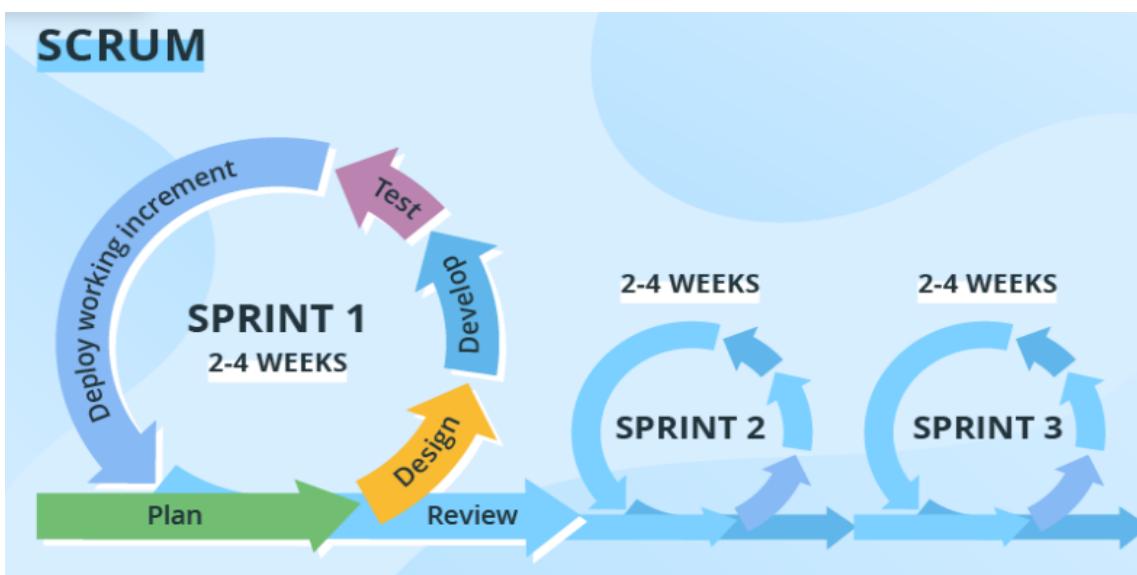


Image: Agile Model

Reference: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-agile-model/>

Advantages:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than processes and tools. Customers, developers, and testers constantly interact with each other.

- Working software is delivered frequently (weeks rather than months).

Disadvantages:

- In the case of some software deliverables, especially large ones, assessing the effort required at the beginning of the software development life cycle is difficult.
- There is a lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is unclear about what final outcome they want.

1.1.4 Getting familiar with terms of Web Development

To understand Web-based application development, you need to understand the terms which are used in web-based software applications. Those terms are web page, static web page, dynamic web page, web site, web application, etc. Let's understand each term in detail.

Web Page

A web page is a document available on the world wide web. Web Pages are stored on a web server and can be viewed using a web browser. A web page can contain huge information including text, graphics, audio, video, and hyperlinks. These hyperlinks are the link to other web pages. A collection of linked web pages on a web server is known as a website. There is a unique Uniform Resource Locator (URL) associated with each web page. A web page is a simple document displayable by a browser. Such documents are written in the HTML language. A web page can embed a variety of different types of resources such as:

- style information — controlling a page's look-and-feel
- scripts — which add interactivity to the page
- media — images, sounds, and videos.

What is Hyperlink?

A hyperlink or simply a link is a selectable element in an electronic document that serves as an access point to other electronic resources. Typically, you click the hyperlink to access the linked resource. Familiar hyperlinks include buttons, icons, image maps, and clickable text links.

All web pages available on the web are reachable through a unique address. To access a page, just type its address in your browser address bar:

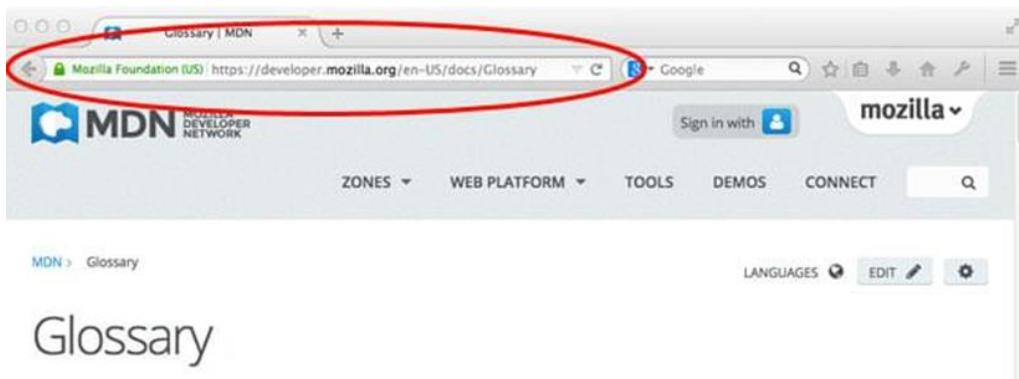


Image: Web Page
 Reference: <https://developer.mozilla.org/enUS/docs/Learn/Commonquestion/Pagesitesserversandsearchengines>

Types of Web Pages

- **Static Web Page:** Static web pages are also known as flat or stationary web pages. They are loaded on the client's browser as exactly they are stored on the web server. Such web pages contain only static information. Users can only read the information but can't do any modifications or interact with the information. Static web pages are created using only HTML. Static web pages are only used when the information is no more required to be modified.

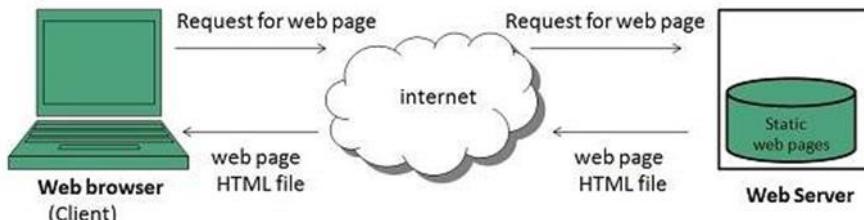


Image: Static Web Page
 Reference: https://www.tutorialspoint.com/internet_technologies/web_pages.html

- **Dynamic Web Page:** Dynamic web page shows different information at different points in time. It is possible to change a portion of a web page without loading the entire web page. It has been made possible using Ajax technology.

✓ Server-side dynamic web page:

It is created by using server-side scripting. There are server-side scripting parameters that determine how to assemble a new web page which also includes setting up more client-side processing.

✓ Client-side dynamic web page:

It is processed using client side scripting such as JavaScript. And then passed in to Document Object Model (DOM).

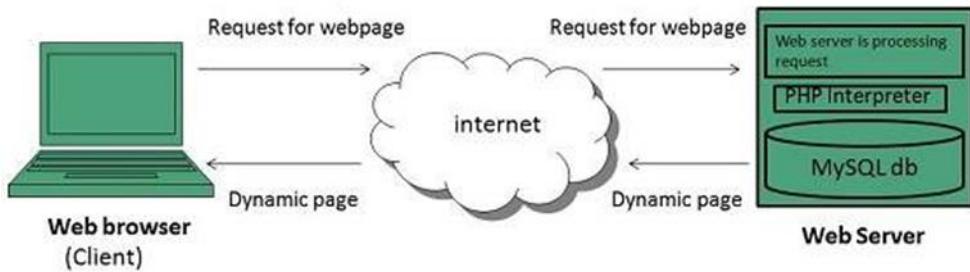


Image: Dynamic Web Page

Reference: https://www.tutorialspoint.com/internet_technologies/web_pages.htm

Web Site

A collection of web pages that are grouped together and usually connected together in various ways. Often called a "website" or a "site". A website is a collection of linked web pages (plus their associated resources) that share a unique domain name. Each web page of a given website provides explicit links—most of the time in the form of a clickable portion of text—that allow the user to move from one page of the website to another. To access a website, type its domain name in your browser address bar, and the browser will display the website's main web page, or homepage (casually referred as "the home").

What is ISP?

ISP stands for Internet Service Provider. They are the companies who provide you with service in terms of internet connection to connect to the internet. You will buy space on a Web Server from any Internet Service Provider. This space will be used to host your Website.

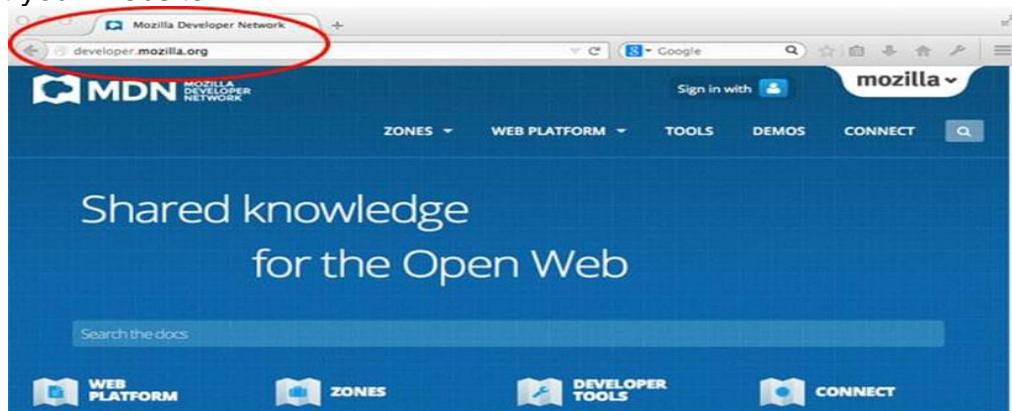


Image: Internet Service Provider

Reference: https://developer.mozilla.org/enUS/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines

The ideas of a web page and a website are especially easy to confuse for a website that contains only one web page. Such a website is sometimes called a single-page website. A website is a collection of publicly accessible, interlinked Web pages that share a single domain name. Websites can be created and maintained by an individual, group, business or organization to serve a variety of purposes.

Together, all publicly accessible websites constitute the World Wide Web. Although it is sometimes called a “web page,” this definition is wrong, since a website consists of several web pages. A website is also known as a “web presence” or simply “site”.

Why do you need a Website?

Here, are prime reasons why you need a website:

- An effective method to showcase your products and services
- Developing a site helps you to create your social proof
- Helps you in branding your business
- Helps you to achieve your business goals
- Allows you to increase your customer support.

Web Application

A web application is a software or program which is accessible using any web browser. Its front end is usually created using languages like HTML, CSS, and Javascript, which are supported by major browsers. While the backend could use any programming stack like LAMP, MEAN, etc. Unlike mobile apps, there is no specific SDK for developing web applications.

Why do you need a Web Application?

- Web applications are more popular because of the following reasons:
- Compared to desktop applications, web applications are easier to maintain as they use the same code in the entire application. There are no compatibility issues.
- Web applications can be used on any platform: Windows, Linux, and Mac... as they all support modern browsers.
- Mobile App store approval is not required in web applications.
- Released at any time and in any form. No need to remind users to update their applications.
- You can access these web applications 24 hours a day and 365 days a year from any PC.
- You can either make use of the computer or your mobile device to access the required data.
- Web applications are a cost-effective option for any organization. Seat Licenses for Desktop software are expensive whereas SaaS, are generally, pays as you go.
- Web-Based Apps are Internet-enabled apps that are accessed through the mobile's web browser. Therefore, you don't require to download or install them.

Here is how a web application works:

- The user creates a request to the web server over the Internet through the application's user interface.
- The web server sends this request to the web application server.

- The web application server executes the requested task, then generates the results of the required data.
- The web application server sends those results back to the web server (requested information or processed data).
- The web server carries the requested information to the client (tablet, mobile device or desktop).
- The requested information appears on the user's display.

Difference Between Web Application and Website

Web Application: A web application is a piece of software that can be accessed by the browser. A Browser is an application that is used to browse the internet. Web application needs authentication. The web application uses a combination of server-side scripts and client-side scripts to present information. It requires a server to manage requests from the users.

Example: Google Apps

Website: The website is a collection of related web pages that contains images, text, audio, video, etc. It can consist of one, two, and n number of pages. A website provides visual and text content that users can view and read. To view a website requires a browser(chrome, firefox). There are many types of websites like Archives, Blog, Community, Dating, etc.

Examples: Amazon, youtube, etc.

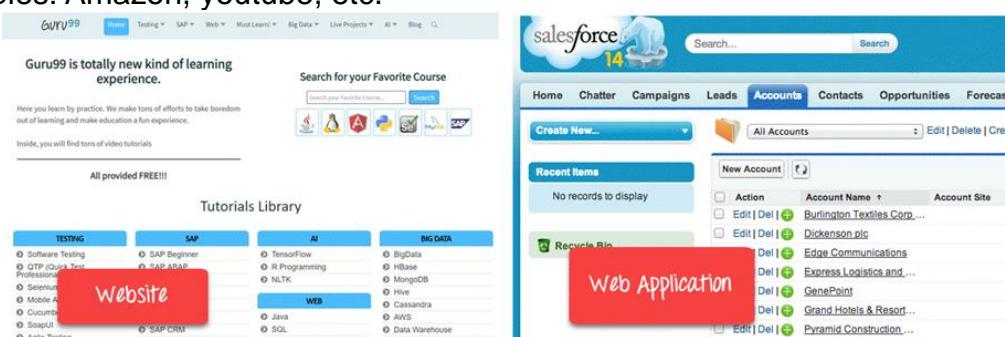


Image: WebSite Vs WebApplication

Reference: <https://www.guru99.com/difference-web-application-website.html>

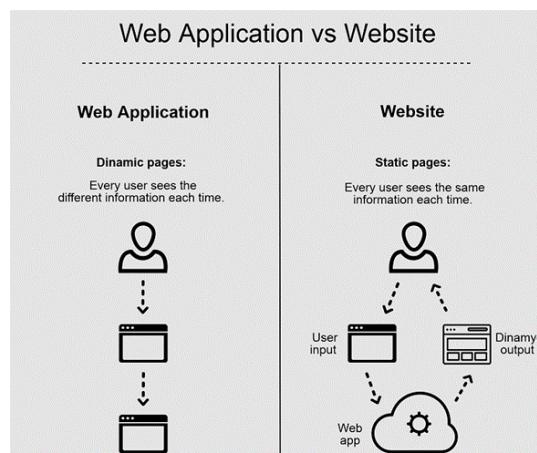


Image: WebSite Vs WebApplication

Reference: <https://www.guru99.com/difference-web-application-website.html>

1.1.5 Web Application Architecture

The architecture of an application describes how its components are interconnected and how they communicate with each other. It can also be described as the connection between the client and server that defines the connection, along with the server that handles the communication between the client and server. It defines how the data is delivered through HTTP and ensures that the client-side server and the backend server can understand. Moreover, it also secures that valid data is present in all user requests. It creates and manages records while providing permission-based access and authentication.

Web Architecture Components

Typically a web-based application architecture comprises 3 core components:

Web Browser: The browser or the client-side component or the front-end component is the key component that interacts with the user, receives the input, and manages the presentation logic while controlling user interactions with the application. User inputs are validated as well if required.

Web Server: The web server also known as the backend component or the server-side component handles the business logic and processes the user requests by routing the requests to the right component and managing the entire application operations. It can run and oversee requests from a wide variety of clients.

Database Server: The database server provides the required data for the application. It handles data-related tasks. In a multi-tiered architecture, database servers can manage business logic with the help of stored procedures.

A 3-tier Architecture

In a traditional 2-Tier architecture, there are two components namely the client side system or the user interface and a backend system which is usually a database server. Here the business logic is incorporated into the user interface or the database server. The downside of 2-tier architecture is that with an increased number of users, the performance decreases. Moreover, the direct interaction of the database and the user device also raises some security concerns.

There are three layers of a 3-Tier architecture:

- Presentation layer / Client Layer
- Application Layer / Business Layer
- Persistence Layer / Data Layer

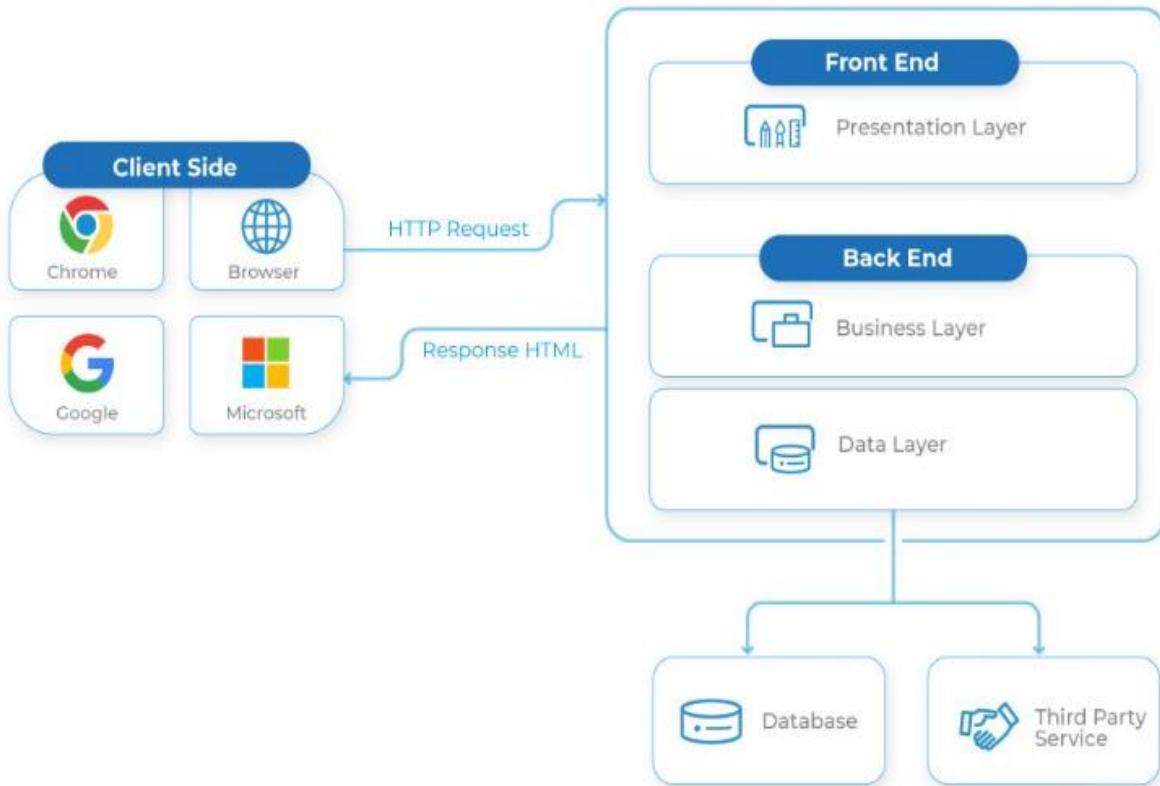


Image: Overview of 3-tier architecture

Reference: <https://www.clickitech.com/devops/web-application-architecture/>

In this model, the intermediate servers receive client requests and process them by coordinating with subordinate servers applying the business logic. The communication between the client and the database is managed by the intermediate application layer thereby enabling clients to access data from different DBMS solutions.

The 3-tier architecture is more secure as the client does not directly access the data. The ability to deploy application servers on multiple machines provides higher scalability, better performance, and better reuse. You can scale it horizontally by scaling each item independently. You can abstract the core business from the database server to efficiently perform load balancing. Data integrity is improved as all data goes through the application server which decides how data should be accessed and by whom. For that reason, a change of management is easy and cost-effective. The client layer can be a thin client which means hardware costs are reduced. This modular model allows you to modify a single tier without affecting the remaining components.

Presentation Layer: This layer is accessible to the client via a browser and it includes user interface components and UI process components. As we have already discussed that these UI components are built with HTML, CSS, and JavaScript (and its frameworks or library) which plays a different role in building the user interface.



Image: Frontend Technologies

Reference: <https://www.clickittech.com/devops/web-application-architecture/>

Business Layer: It is also referred to as a Business Logic or Domain Logic or Application Layer. It accepts the user's request from the browser, processes it, and regulates the routes through which the data will be accessed. The whole workflow is encoded in this layer. You can take the example of booking a hotel on a website. A traveler will go through a sequence of events to book the hotel room and the whole workflow will be taken care of by the business logic.



Image: Backend Technologies

Reference: <https://www.clickittech.com/devops/web-application-architecture/>

Persistence Layer: It is also referred to as a storage or data access layer. This layer collects all the data calls and provides access to the persistent storage of an application. The business layer is closely attached to the persistence layer, so the logic knows which database to talk to and the process of retrieving data becomes more optimized. A server and a database management system software exist in the data storage infrastructure which is used to communicate with the database itself, applications, and user interfaces to retrieve data and parse it. You can store the data in hardware servers or in the cloud.

1.2 Building Blocks of Frontend Web Application Development

The client-side component of a web application architecture enables users to interact with the server and the backend service via a browser. The code resides in the browser, receives requests, and presents the user with the required information. This is where UI/UX design, dashboards, notifications, configurational settings, layout, and interactive elements come into the picture.

Here are some of the commonly used front-end technologies:

HTML: HTML or Hypertext Markup Language is a popular standard markup language that enables developers to structure web page contents using a series of page elements. Developed by Tim Berners-Lee and released in 1993, HTML quickly evolved and became the standard markup language across the globe.

CSS: CSS or Cascading Style Sheets is a popular style sheet language that enables developers to separate website content and layout for sites developed using markup languages. Using CSS, you can define a style for elements and reuse them multiple times. Similarly, you can apply one style across multiple sites. It is simple and easy to learn. You can apply a style for a single element, an entire webpage, or the entire website. It is device-friendly too.

Browser compatibility and security are two areas that raise a concern. Similarly, different versions of CSS also create confusion. It is advised for developers to check the compatibility before making any changes to the design.

JavaScript: JavaScript or JS is the most popular client-side programming language which is used by more than 90% of websites in recent times. It was designed by Brendan Eich of Netscape in 1995. JavaScript uses a simple, easy-to-learn syntax. The language is so popular that every browser comes with a JS engine to run JavaScript code on devices. It is easy to insert JS code on any web page which makes it highly interoperable. It allows you to create rich interfaces to deliver a better UI/UX experience. Being on the client side, JS reduces the server load as well.

Let's learn each of mentioned above in detail in the next section.

1.2.1 Getting Started with HTML

HTML stands for Hyper Text Markup Language, is the standard markup language used to create web pages. Along with CSS, and JavaScript, HTML is a cornerstone technology used to create web pages, as well as to create user interfaces for mobile and web applications.

- HTML stands for Hyper Text Markup Language
- HTML elements are the building blocks of HTML pages
- HTML describes the structure of Web pages using HyperText markup language
- HTML elements are represented by tags
- When you test your page on browsers. Browsers do not display the HTML tags but use them to render the content of the page.
- HTML page extension always will be .html (example.html)

Features of HTML:

- It is easy to learn and easy to use.
- It is platform-independent.
- Images, video, and audio can be added to a web page.
- Hypertext can be added to the text.
- It is a markup language.

Advantages:

- HTML is used to build a website.
- It is supported by all browsers.
- It can be integrated with other languages like CSS, JavaScript, etc.

Disadvantages:

- HTML can create only static webpages so for dynamic web pages other languages have to be used.
- A large amount of code has to be written to create a simple web page.
- The security feature is not good.

Creating HTML Document:

Creating an HTML document is easy. To begin coding HTML you need only two things: a simple text editor and a web browser. Notepad is the most basic of simple-text editors and you will probably code a fair amount of HTML with it.

- Open Notepad or another text editor.
- At the top of the page type <html>.
- On the next line, indent spaces and now add the opening header tag: <head>.
- On the next line, indent ten spaces and type <title> </title>.
- Go to the next line, indent five spaces from the margin, and insert the closing header tag: </head>.
- Five spaces in from the margin on the next line, type <body>.
- Now drop down another line and type the closing tag right below its mate: </body>.
- Finally, go to the next line and type </html>.

- In the File menu, choose Save As.
- In the Save as Type option box, choose All Files.
- Name the file template.html.
- Click Save.

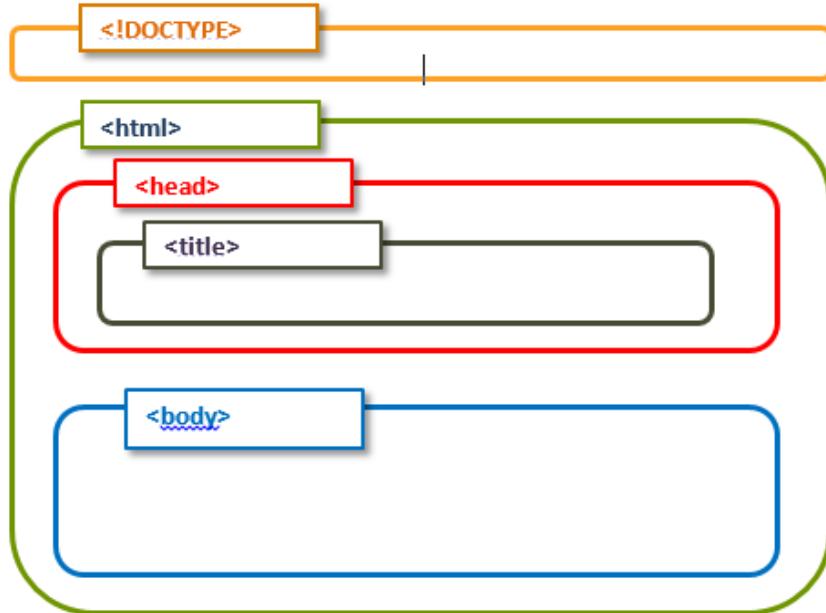


Image: HTML Page Structure

Reference: <https://qatechhub.com/html-page-structure-nesting/>

<DOCTYPE! html>: This tag is used to tell the HTML version. This currently tells that the version is HTML 5.

<html>: This is called the HTML root element and is used to wrap all the code.

<head>: Head tag contains metadata, title, page CSS etc. All the HTML elements that can be used inside the <head> element are: <style> ,<title> ,<base> ,<noscript> ,<script> ,<meta> ,<title> ,etc.

<body>: The body tag is used to enclose all the data that a web page has from texts to links. All of the content that you see rendered in the browser is contained within this element.

Example Code:

```

<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
  
```

Output:



HTML Formatting Tags

HTML also defines special elements for defining text with a special meaning. The formatting tags are divided into two categories: physical tags, used for text styling (visual appearance of the text), and logical or semantic tags used to add semantic value to the parts of the text (for example, let the search engines know what keywords should be web page ranked for).

HTML uses elements like **** and **<i>** for formatting output, like bold or italic text. Formatting elements were designed to display special types of text:

**** - Bold text
**** - Important text
<i> - Italic text
**** - Emphasized text
<mark> - Marked text
<small> - Small text
**** - Deleted text
<ins> - Inserted text
<sub> - Subscript text
<sup> - Superscript text

**** and **** Tags

The **** tag is a physical tag that stands for bold text, whereas the **** tag being a logical tag is used to emphasize the importance of the text.

****This text is bold****
****This text is strong****

<i> and **** Tags

The **<i>** and **** tags define italic text. The **<i>** tag is only responsible for visual appearance of the enclosed text, without any extra importance. The **** tag defines emphasized text, with added semantic importance.

<i>This text is italic**</i>**
****This text is emphasized****

<pre> Tag

The `<pre>` tag is used to define preformatted text. The browsers render the enclosed text with white spaces and line breaks.

<pre>Spaces

```
    and line breaks  
    within this element  
    are shown as typed.  
</pre>
```

<mark> Tag

The `<mark>` tag is used to present a part of text in one document as marked or highlighted for reference purposes.

<small> Tag

The `<small>` tag decreases the text font size by one size smaller than a document's base font size (from medium to small, or from x-large to large).

The tag usually contains items of secondary importance such as copyright notices, side comments, or legal notices.

```
<p>The interest rate is only 10%*</p>  
<small>* per day</small>
```

<big> Tag

The `<big>` tag defines bigger text.

```
<p><big>Bigger text</big></p>
```

** and <s> Tags**

The `` tag specifies a part of the text that was deleted from the document. Browsers display this text as a strikethrough.

```
<p> She likes <del>violets</del> snowdrops.</p>  
<p><s>I am studying in high school.</s></p>
```

The `<s>` tag defines text that is no longer correct, or relevant.

The content of both tags is displayed as strikethrough. However, despite the visual similarity, these tags cannot replace each other.

<ins> and <u> Tags

The `<ins>` tag defines the text that has been inserted (added) to the document. The content of the tag is displayed as underlined.

```
<p>She likes <del>violets</del> <ins>snowdrops</ins>.</p>
```

The `<u>` tag specifies text that is stylistically different from normal text, i.e. words or fragments of text that need to be presented differently. This could be misspelled words or proper nouns in Chinese.

```
<p>Here we used <u>the &lt;u&gt; tag</u>.</p>
```

<sub> and <sup> Tag

The `<sub>` defines subscript texts. Subscript text is under the baseline of other symbols of the line and has a smaller font. The `<sup>` tag defines superscript, which is set slightly above the normal line of type and is relatively smaller than the rest of the text. The baseline passes through the upper or lower edge of the symbols.

<p>The formula of water is H₂O, and the formula of alcohol is C₂H₅OH </p>

<p>E = mc², where E — kinetic energy, m — mass, c — the speed of light. </p>

<dfn> Tag

The `<dfn>` tag is used to define the term, that is mentioned for the first time. The content of the tag is displayed in italic.

<p><dfn>HTML</dfn> (HyperText Markup Language) — The standardized markup language for documents on the World Wide Web. Most web pages contain a description of the markup in the language HTML</p>

**<p>,
 and <hr> Tags**

The `<p>` tag defines the paragraph. Browsers automatically add 1em margin before and after each paragraph.

<p>The first paragraph</p>

The `
` tag inserts a single line break. Unlike the `<p>` tag an empty indent is not added before the line.

<h1>How to use the
 tag</h1>

<p> We can insert the
 tag inside the paragraph,
 to transfer a part of the text to another line if necessary.</p>

In HTML5 the `<hr>` tag defines a thematic change between paragraph-level elements in an HTML page. In previous versions of HTML, it was used to draw a horizontal line on the page visually separating the content. In HTML5 the element has semantic meaning.

<h1>Football</h1>

<p>Team sports involve, to varying degrees, kicking a ball with a foot to score a goal.</p>

<hr>

<h1>Basketball</h1>

<p>A game played between two teams of five players in which goals are scored by throwing a ball through a netted hoop fixed at each end of the court.</p>

Note: Some formatting tags like ``, `<u>`, `<small>`, `<big>`, `<i>`, `<mark>`, `<ins>`, ``, etc are not used as the same properties are added as a CSS property in CSS3 and HTML5. Initially, in HTML4 we were utilizing these tags but after adding the same properties in CSS, avoid using these tags while designing web pages.

Difference between HTML4 and HTML5

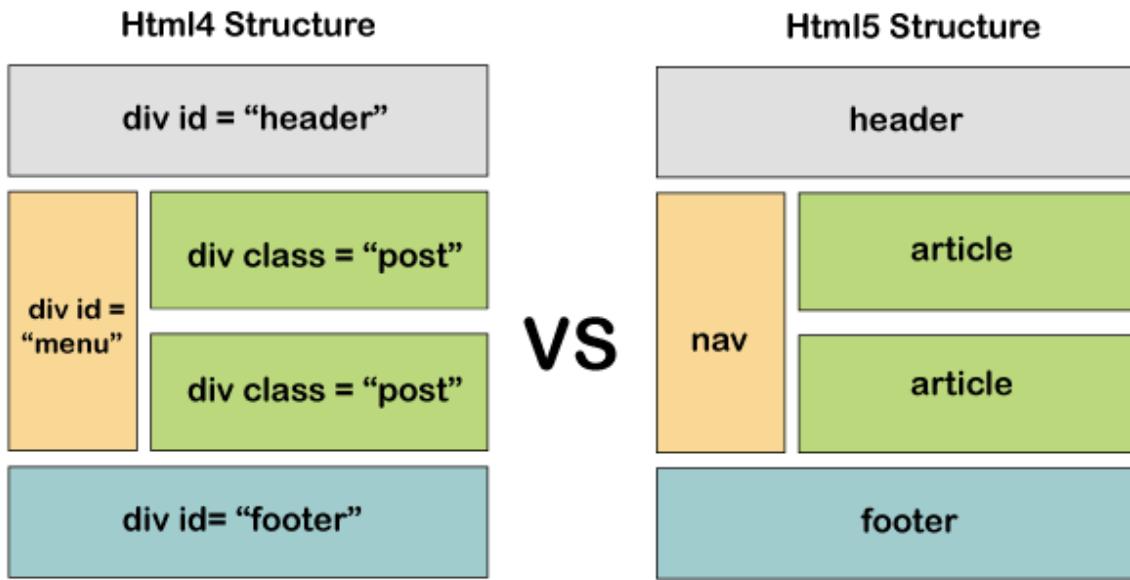


Image: Web Page layout difference between HTML4 & HTML5

Reference: <https://www.javatpoint.com/html-vs-html5>

HTML is referred to as the primary language of the World Wide Web. HTML has had many updates over time, and the latest HTML version is HTML5. There are some differences between the two versions:

- HTML5 supports both audio and video while none of them were part of HTML.
- HTML cannot allow JavaScript to run within the web browser, while HTML5 provides full support for running JavaScript.
- In HTML5, inline MathML and SVG can be used in a text, while in HTML it is not possible.
- HTML5 supports new types of form controls, such as date and time, email, number, category, title, Url, search, etc.
- Many elements have been introduced in HTML5. Some of the most important are time, audio, description, embed, fig, shape, footer, article, canvas, navy, output, section, source, track, video, etc.

Features	Html	Html5
definition	A hypertext markup language (HTML) is the primary language for developing web pages.	HTML5 is a new version of HTML with new functionalities with markup language with Internet technologies.
Multimedia support	Language in HTML does not have support for video and audio.	HTML5 supports both video and audio.
Storage	The HTML browser uses cache memory as temporary storage.	HTML5 has the storage options like: application cache, SQL database, and web storage.
Browser	HTML is compatible with almost all	In HTML5, we have many

compatibility	browsers because it has been present for a long time, and the browser made modifications to support all the features.	new tags, elements, and some tags that have been removed/modified, so only some browsers are fully compatible with HTML5.
Graphics support	In HTML, vector graphics are possible with tools LikeSilver light, Adobe Flash, VML, etc.	In HTML5, vector graphics are supported by default.
Threading	In HTML, the browser interface and JavaScript running in the same thread.	The HTML5 has the JavaScript Web Worker API, which allows the browser interface to run in multiple threads.
Storage	Uses cookies to store data.	Uses local storage instead of cookies
Vector and Graphics	Vector graphics are possible with the help of technologies like VML, Silverlight, Flash,etc.	Vector graphics is an integral part of HTML5, SVG and canvas.
Shapes	It is not possible to create shapes like circles, rectangles, triangles.	We can draw shapes like circles, rectangles, triangles.
Doc type	Doctype declaration in html is too long <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	The DOCTYPE declaration in html5 is very simple "<!DOCTYPE html>"
Character Encoding	Character encoding in HTML is too long. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">	Character encoding declaration is simple <meta charset = "UTF-8">
Multimedia support	Audio and video are not part of HTML4.	Audio and video are essential parts of HTML5, like <Audio>, <Video>.
Vector Graphics	In HTML4, vector graphics are possible with the help of techniques like VML, Silver light, and Flash.	Vector graphics are an integral part of HTML5, SVG, and Canvas.
Shapes	It is not possible to draw shapes like circles, rectangles, or triangles.	Using html5, you can draw shapes like circles, rectangles, and triangles.
Browser Support	Works with all older browsers	A new browser supports this.

HTML Block Component

HTML Blocks are grouping items of HTML elements. Each HTML element has its own display based on its type. However, HTML blocks also have display but can group the other HTML elements also as block.

By default, all block-level elements start on a new line. The elements added after block-level elements will start from a new line. Block-level elements can contain other block-line elements as nested block-line elements. Block-level elements can stretch to the browser's full width.

Below are some of the block-level elements:

- <address> - Specifies the contact information for a page or document
- <form> - A Facility to submit the input data by user
- <blockquote> - used to define a quote section
- <p> - Paragraph
- <pre> - Preformatted text in HTML document
- <h1> to <h6> - Used to define HTML headings
- <dd> - Used to specify the term/description/name in a description list
- <div> - used to create block-level elements
- <table> - represents a table in an HTML document
- - Ordered list in HTML documents
- - Unordered list in HTML document
- <nav> - Specifies the navigation area

Block elements can be divided into two types.

- Inline Elements
- Group Elements

Inline elements

The inline elements do not start at the newline. The inline elements always start in the middle of the line and are part of another element. The below list is the inline elements.

- - used to display the text as bold
- <i> - Alternate Voice or different quality of text
- <u> - Unarticulated text/underlined text in HTML document
- <p> - Paragraph
- <address> - Specifies the contact information for a page or document
- <form> - A Facility to submit the input data by user
- - used to represent a list item in HTML document
- <ins> - Editorial Insertion
- - Used for editorial deletion
- <code> - Used to define the computer code

Group elements

The Group elements always start with new line. The Group elements are used to group other elements. <div> and are the majorly used Group elements.

<div>: <div> is mostly used to create block-level elements. In other words, <div> tag is used to define a section in HTML documents. Except <div>, all others have their own display. The tag can be coded like <div></div> with its contents/other HTML tags inserted between the opening and closing tags. <div> also acts as a block-level element that is used as a container for other HTML elements. The <div> element has

no additional attributes by default. style and class are commonly provided with DIV elements. When `<div>` element is used together with CSS, the style blocks can be used together. All browsers always place a line break before and after the `<div>` tag by default.

Syntax -

```
<div>.....</div>
```

****: Generic container for text or group inline-elements HTML. The `` tag used to specify the generic container for inline elements and content. There no visual change that can be done by `` tag. It doesn't represent anything other than its children. The tag can be specified like `` with the content between the opening and closing tags.

Syntax -

```
<span>.. text here.. </span>
```

List

HTML List Tags are used to specify information in the form of a list. HTML Lists are very useful to group related information together. Often List items look well-structured and they are easy to read for users. A list can contain one or more list elements.

HTML List Types		
List Type	Description	Tags used
Unordered List	used to group a set of items without any order.	<code>,</code>
Ordered List	used to group a set of items, in a specific order.	<code>,</code>
Definition List	used to display some definition term (<code>dt</code>) & definition's description (<code>dd</code>)	<code><dl>,<dt>,<dd></code>

Unordered List: Unordered lists are used to list a set of items when they have no special order or sequence. It is also called a bulleted list. An unordered list is created using HTML `` tag. Each item in the list starts with the `` tag

Example Code:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Output:

- Coffee
- Tea
- Milk

HTML List Types		
List Style Type	Description	Example & Syntax
disc	Starts a list using discs type bullets (default)	<ul type="disc">
circle	Starts a list using circle-type bullets	<ul type="circle">
square	Starts a list using square-type bullets	<ul type="square">
none	Starts a list without bullets	<ul type="none">

Ordered List: Ordered list is used to list related items in a numbered or other specific order. This is useful when you want to show counts of items in some way. The ordered list is created using HTML tag. Each item in the list is with the tag.

Example Code:

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Output:

- 1. Coffee
- 2. Tea
- 3. Milk

HTML Ordered List Style Type Attribute		
List Style Type	Description	Example and Syntax
Numbers	Starts a list using numbers (default)	<ol type="1">
Uppercase letters	Starts a list using uppercase letters	<ol type="A">
Lowercase letters	Starts a list using lowercase letters	<ol type="a">
Uppercase roman numbers	Starts a list using uppercase roman numbers	<ol type="I">
Lowercase roman numbers	Starts a list using lowercase roman numbers	<ol type="i">

Definition List or Description List: The definition list is different from the other two types of lists. No bullet or number is provided for the list of items. In this list type, the list element has two parts.

(1) A definition term and (2) The definition description

The definition list is surrounded within <DL> </DL> tags.

Definition term is presented in between <DT> </DT> tag and

Definition description should be surrounded within <DD> </DD> tag.

Example Code:

```
<dl>
<dt>Coffee</dt>
<dd>- black hot drink</dd>
<dt>Milk</dt>
<dd>- white cold drink</dd>
</dl>
```

Output:

```
Coffee
- black hot drink
Milk
- white cold drink
```

Table

You can create tables for your website using the `<table>` tag in conjunction with the `<tr>`, `<td>` and `<th>` tags. The HTML tables allow displaying the data (e.g. image, text, link) in columns and rows of cells. Table rows may be grouped into head, foot, and body sections (via the `<thead>`, `<tfoot>`, and `<tbody>` elements, respectively).

The `<caption>` tag defines a table caption. A `<caption>` tag must be inserted immediately after the `<table>` tag.

Note: You can specify only one caption per table.

`cellpadding` - The cell padding attribute places spacing around data within each cell.

`cellspacing` - The cell spacing attribute places space around each cell in the table

th	- Table Head
tr	- Table Row
td	- Table Data

Example Code:

```
<table border=2>
<tr>
<th>Company</th>
<th>Contact</th>
<th>Country</th>
</tr>
<tr>
<td>Alfreds Futterkiste</td>
<td>Maria Anders</td>
<td>Germany</td>
</tr>
<tr>
<td>Centro comercial Moctezuma</td>
<td>Francisco Chang</td>
<td>Mexico</td>
</tr>
```

</table>

Output:

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico

Link

HTML links are hyperlinks. You can click on a link and jump to another document. The HTML tag defines a hyperlink. The most important attribute of the element is the href attribute, which indicates the link's destination. The link text is the part that will be visible to the reader. Clicking on the link text will send the reader to the specified URL address.

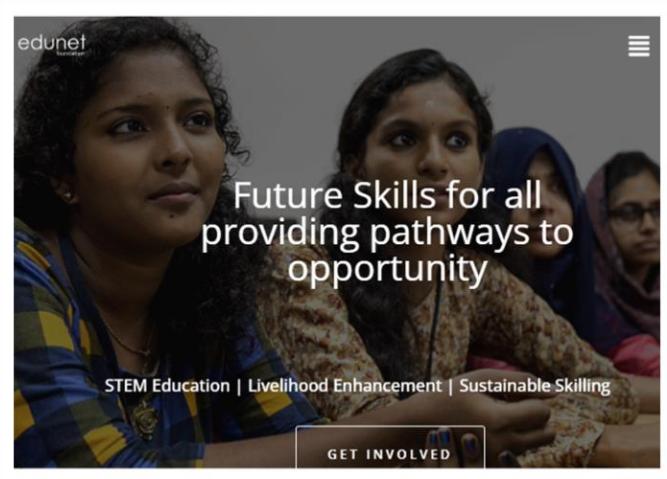
Example Code:

```
<a href="https://www.edunetfoundation.org/">Visit Edunet Foundation!</a>
```

Output:

[Visit Edunet Foundation!](https://www.edunetfoundation.org/)

When you click on the link it will showcase the below output on the browser screen



Form

HTML Forms are required to collect different kinds of user inputs, such as contact details like name, email address, phone numbers, or details like credit card information, etc.

Forms contain special elements called controls like input box, checkboxes, radio buttons, submit buttons, etc. Users generally complete a form by modifying its

controls e.g., entering text, selecting items, etc., and submitting this form to a web server for further processing.

Action Attribute: The action attribute defines the action to be performed when the form is submitted. Usually, the form data is sent to a page on the server when the user clicks on the submit button.

Example : <form action="/action_page.php">

In the example above, the form data is sent to a page on the server called "/action_page.php". This page contains a server-side script that handles the form data:

If the action attribute is omitted, the action is set to the current page.

Target Attribute: The target attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window. The default value is "_self" which means the form will be submitted in the current window. To make the form result open in a new browser tab, use the value "_blank".

Here, the submitted result will open in a new browser tab:

```
<form action="/action_page.php" target="_blank">
```

Method Attribute: The method attribute specifies the HTTP method (GET or POST) to be used when submitting the form data.

```
<form action="/action_page.php" method="get">  
<form action="/action_page.php" method="post">
```

Label Element: The <label> tag defines a label for many form elements.

The <label> element is useful for screen-reader users because the screen-reader will read out and load the label when the user is focused on the input element.

Input Element: This is the most commonly used element within HTML forms. It allows you to specify various types of user input fields, depending on the type attribute. An input element can be of type text field, password field, checkbox, radio button, submit button, reset button, file select box, as well as several new input types introduced in HTML5.

<input> element can be displayed in several ways, depending on the type attribute. If the type attribute is omitted, the input field gets the default type: "text".

Text Fields: Text fields are one line areas that allow the user to input text.

Example Code:

```
<form> <label>User Name:</label>  
<input type="text" name="username" id="username"> </form>
```

Output:

User Name:

Password Field: Password fields are similar to text fields. The only difference is; characters in a password field are masked, i.e. they are shown as asterisks or dots. This is to prevent someone else from reading the password on the screen.

Example Code:

```
<form> <label>Password:</label>
<input type="password" name="user-password" id="user-pwd"> </form>
```

Output:

Password:

Radio Button Field: Radio buttons let a user select ONLY ONE of a limited number of choices.

Example Code:

```
<form>
<input type="radio" id="html" name="fav_language" value="HTML">
<label>HTML</label><br>
<input type="radio" id="css" name="fav_language" value="CSS">
<label>CSS</label><br>
<input type="radio" id="javascript" name="fav_language" value="JavaScript">
<label>JavaScript</label><br><br>
</form>
```

Output:

- HTML
- CSS
- JavaScript

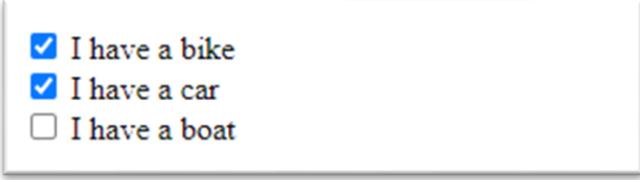
Check box Field: Checkboxes let a user select ZERO or MORE options of a limited number of choices

Example Code:

```
<form>
<input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
<label> I have a bike</label><br>
<input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
<label> I have a car</label><br>
<input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
```

```
<label> I have a boat</label><br><br>
</form>
```

Output:

- 
- I have a bike
 - I have a car
 - I have a boat

Other input types:

Button – to create button

Color - a color picker can show up

Date - a date picker can show up

Email - the e-mail address can be automatically validated when submitted

Image - defines an image as a submit button

File - defines a file-select field and a "Browse" button for file uploads

Hidden - defines a hidden input field (not visible to a user)

Month - allows the user to select a month and year

Number - defines a numeric input field

Range - defines a control for entering a number whose exact value is not important

Submit – used to create submit button

Reset – used to create reset button

Attribute	Description
checked	Specifies that an input field should be pre-selected when the page loads (for type="checkbox" or type="radio")
disabled	Specifies that an input field should be disabled
max	Specifies the maximum value for an input field
maxlength	Specifies the maximum number of characters for an input field
min	Specifies the minimum value for an input field
pattern	Specifies a regular expression to check the input value against
readonly	Specifies that an input field is read-only (cannot be changed)
required	Specifies that an input field is required (must be filled out)
size	Specifies the width (in characters) of an input field
step	Specifies the legal number intervals for an input field
value	Specifies the default value for an input field

Drop down Field: <select> and <option> tags are used to create a drop down list in html form.

The <select> element defines a drop-down list

The <option> elements define an option that can be selected.

By default, the first item in the drop-down list is selected.

To define a pre-selected option, add the selected attribute to the option

Example Code:

```
<form>
  <label for="cars">Choose a car:</label>
  <select id="cars" name="cars">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
</form>
```

Output:

Choose a car: **Volvo** ▾

Multi-Media

What is Multimedia?

Multimedia comes in many different formats. It can be almost anything you can hear or see, like images, music, sound, videos, records, films, animations, and more. Web pages often contain multimedia elements of different types and formats.

Browser Support

The first web browsers had support for text only, limited to a single font in a single color. Later came browsers with support for colors, fonts, images, and multimedia!

Multimedia Formats

Multimedia elements (like audio or video) are stored in media files. The most common way to discover the type of a file, is to look at the file extension. Multimedia files have formats and different extensions like: .wav, .mp3, .mp4, .mpg, .wmv, and .a



Image: HTML Audio & Video Format

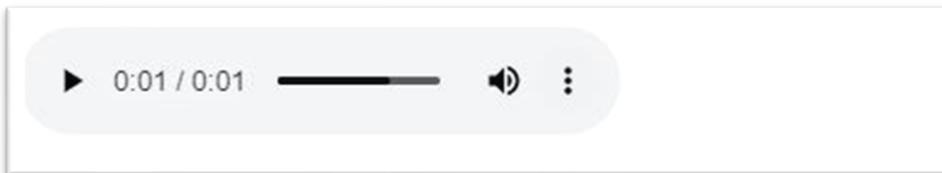
Reference: <https://www.sitesbay.com/html5/html5-audio-video-formats>

The HTML5 `<audio>` element has specified a standard method to embed audio in a web page.

Example Code:

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

Output:



The `controls` attribute is used to add audio controls, like play, pause, and volume. Text between the `<audio>` and `</audio>` tags will be displayed in browsers which does not support the `<audio>` element.

Multiple `<source>` elements can be linked to different audio files. The browser will be using the first recognized format.

The HTML5 `<video>` element is used to specify a standard way of embedding a video in a web page.

Example Code:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogv" type="video/ogv">
Your browser does not support the video tag.
</video>
```

Output:



The `controls` attribute is used to add video controls, like play, pause, and volume. It is a good method to always use width and height attributes. If height and width are not mentioned, the browser will not know the size of the video. The effect of this is that page will be changing (or flicking) while the video gets loaded. Text between the

<video> and </video> tags will only get displayed in browsers that do not support the <video> element. Multiple <source> elements can be used to link different video files. The browser will be using the first recognized format.

Graphics

Web graphics are visual representations used on a Web site to enhance or enable the representation of an idea or feeling, in order to reach the Web site user. Graphics may entertain, educate, or emotionally impact the user, and are crucial to strength of branding, clarity of illustration, and ease of use for interfaces. Examples of graphics include maps, photographs, designs and patterns, family trees, diagrams, architectural or engineering blueprints, bar charts and pie charts, typography, schematics, line art, flowcharts, and many other image forms.

Graphic designers have many tools and technologies at their disposal for everything from print to Web development, and W3C provides many of the underlying formats that can be used for the creation of content on the open Web platform.

What is Graphics Used For?

Graphics are used for everything from enhancing the appearance of Web pages to serving as the presentation and user interaction layer for full-fledged Web Applications. Different use cases for graphics demand different solutions, thus there are several different technologies available. Photographs are best represented with PNG, while interactive line art, data visualization, and even user interfaces need the power of SVG and the Canvas API. CSS exists to enhance other formats like HTML or SVG. WebCGM meets the needs for technical illustration and documentation in many industries.

Graphic Elements in HTML5

- Canvas
- Scalable Vector Graphics (SVG)

Canvas Element:

- helps the browser to draw shapes and images without any plugin.
- is used to draw graphics, on the fly, via scripting.
- has several methods for drawing paths, boxes, circles, characters and adding images.

The following is a sample of drawing a rectangle using a Canvas Element:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5      <script type="text/javascript">
6
7          function SetCanvas()
8          {
9              var can = document.getElementById("canvas1");
10             var ctx = can.getContext("2d");
11
12             ctx.fillStyle="#ff0000";
13             ctx.strokeStyle = "Black"
14             ctx.lineWidth=2;
15
16             ctx.fillRect(0,0,100,200);
17             ctx.strokeRect(0,0,100,200);
18         }
19
20     </script>
21
22 </head>
23 <body onload="SetCanvas()">
24
25     <canvas id="canvas1" width="1000" height="1000" >Browser Not Supported</canvas>
26
27 </body>
28 </html>
29

```

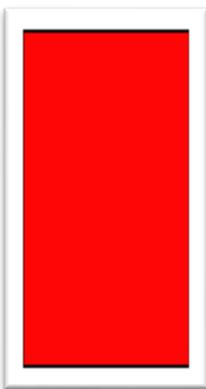
Get Object and Context

Set Properties

Call Methods to Fill

Define Canvas Object

Output:



The following is a sample of drawing a circle using a Canvas Element:

```

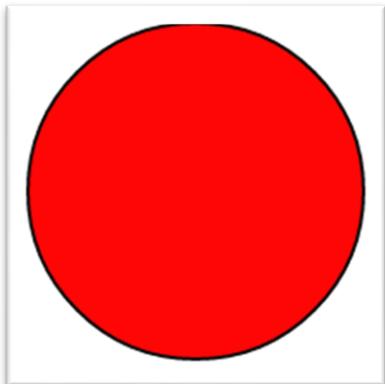
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title></title>
5   <script type="text/javascript">
6
7     function SetCanvas()
8     {
9       var can = document.getElementById("canvas1");
10      var ctx = can.getContext("2d");
11
12      ctx.fillStyle="#ff0000";
13      ctx.strokeStyle = "Black"
14      ctx.lineWidth=2;
15
16
17      ctx.arc(200,200,100,0,2* Math.PI);
18      ctx.fill();
19      ctx.stroke();
20
21    }
22
23 </script>
24
25 </head>
26 <body onload="SetCanvas();>
27   <canvas id="canvas1" width="1000" height="1000" >Browser Not Supported</canvas>
28
29 </body>
30 </html>
31
32

```

Diagram illustrating the steps in the JavaScript code:

- Get Object and Context**: Points to the line `var ctx = can.getContext("2d");`.
- Set Properties**: Points to the lines `ctx.fillStyle="#ff0000";` and `ctx.strokeStyle = "Black";`.
- Call methods to fill**: Points to the lines `ctx.arc(200,200,100,0,2* Math.PI);`, `ctx.fill();`, and `ctx.stroke();`.
- Define Canvas Object**: Points to the line `<canvas id="canvas1" width="1000" height="1000" >Browser Not Supported</canvas>`.

Output:



Scalable Vector Graphics (SVG)

- is based on a vector-based family of graphics.
- defines graphics in XML Format.
- creates graphics that does not lose any quality when zoomed or resized

The following is a sample of drawing a rectangle using an <SVG> Element:

```

1  <!DOCTYPE HTML>
2  <html>
3  <head>
4
5  <title>Scalar Vector Graphics
6  </title>
7  </head>
8
9  <body>
10
11 <svg height="500" width="500" >
12   <rect x="50" y="100" height="100" width = "200" fill="red" stroke="black" stroke-width="2" />
13 </svg>
14
15 </body>
16
17 </html>

```

This create object with inbuilt properties. No Need for any Java Script

Output:



The following is a sample of drawing a circle using an SVG Element:

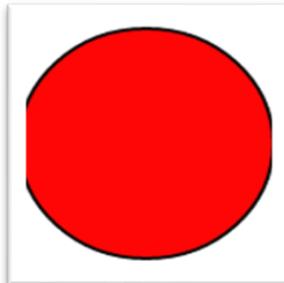
```

1  <!DOCTYPE HTML>
2  <html>
3  <head>
4
5  <title>Scalar Vector Graphics
6  </title>
7  </head>
8
9  <body>
10
11 <svg height="500" width="500">
12   <circle cx="100" cy="100" r="70" fill="red" stroke="black" stroke-width="2" />
13 </svg>
14
15 </body>
16
17 </html>

```

This create object with inbuilt properties.No Need for any java script.

Output:



Differences between Canvas and SVG elements

Canvas	SVG
Canvas draws 2D graphics, on the fly (with a JavaScript).	SVG defines the graphics in XML format.
Resolution dependent.	Resolution independent.
Canvas is rendered pixel by pixel.	In SVG, each drawn shape is remembered as an object.

Iframe

An HTML iframe is used to display a web page within a web page. The HTML `<iframe>` tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document.

Example Code:

```
<iframe src="demo_iframe.html" height="200" width="300" title="Iframe Example"></iframe>
```

Output:



Layout Elements

HTML has several semantic elements that define the different parts of a web page:

- <header> - Defines a header for a document or a section
- <nav> - Defines a set of navigation links
- <section> - Defines a section in a document
- <article> - Defines an independent, self-contained content

<aside> - Defines content aside from the content (like a sidebar)
<footer> - Defines a footer for a document or a section
<details> - Defines additional details that the user can open and close on demand
<summary> - Defines a heading for the <details> element

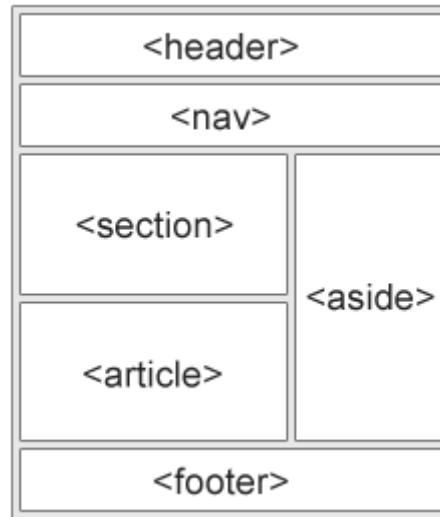


Image: HTML Page Layout using semantic elements
Reference: https://www.w3schools.com/html/html_layout.asp

Geolocation API

The HTML Geolocation API is used to get the geographical position of a user. Since this can compromise privacy, the position is not available unless the user approves it. The `getCurrentPosition()` method is used to return the user's position.

Example Code:

```
<html>
<body>

<p>Click the button to get your coordinates.</p>

<button onclick="getLocation()">Try It</button>

<p id="demo"></p>

<script>
var x = document.getElementById("demo");

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}
```

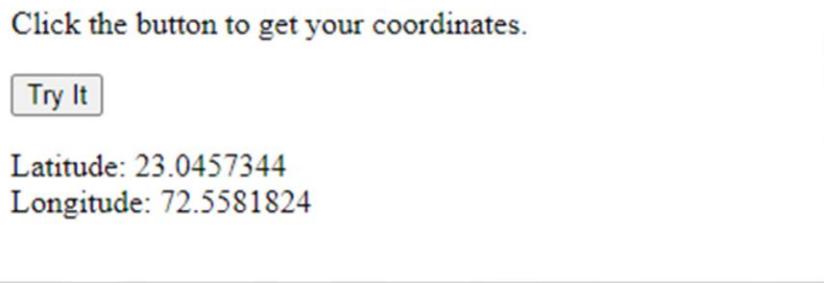
```

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>

</body>
</html>

```

Output:



Click the button to get your coordinates.

Try It

Latitude: 23.0457344
Longitude: 72.5581824

1.2.2 Styling with CSS

What is CSS?

CSS (Cascading Style Sheets) is used to style web pages. Cascading Style Sheets are fondly referred to as CSS. The reason for using this is to simplify the process of making web pages presentable. It allows you to apply styles on web pages. More importantly, it enables you to do this independently of the HTML that makes up each web page.

Why we learn CSS?

Styling is an essential property for any website. It increases the standards and overall look of the website that makes it easier for the user to interact with it. A website can be made without CSS, as styling is MUST since no user would want to interact with a dull and shabby website. So, for knowing Web Development, learning CSS is mandatory.

CSS Code Format: It is the basic structure of the CSS style of a webpage.

```

body {
  background-color: lightgray;
}

h1 {
  color: green;
  text-align: center;
}

p {
  font-family: sans-serif;
  font-size: 16px;
}

```

```

<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: lightblue;
}

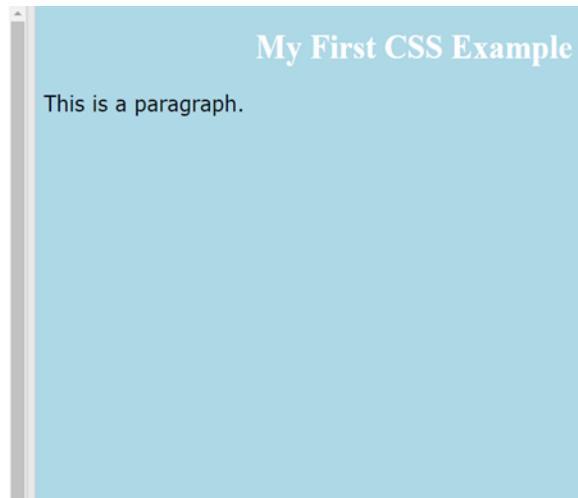
h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
</style>
</head>
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>

```



Types of CSS: There are three types of CSS which are given below.

- **Inline:** Inline CSS contains the CSS property in the body section attached with the element known as inline CSS.
- **Internal or Embedded:** The CSS ruleset should be within the HTML file in the head section i.e the CSS is embedded within the HTML file.
- **External:** External CSS contains a separate CSS file that contains only style properties with the help of tag attributes.

External CSS: External CSS contains a separate CSS file with a .css extension which contains only style property with the help of tag attributes.

```

selector{
    property1: value1;
    property2: value2;
}

```

Include external CSS file: The external CSS file is linked to the HTML document using a link tag.

```
<link rel="stylesheet" type="text/css" href="/style.css" />
```

Internal CSS or Embedded: CSS is embedded within the HTML file using a style HTML tag.

```

<style type="text/css">
div { color: #444;}
</style>

```

Inline CSS: It contains CSS properties in the body section specified within HTML tags.

```
<tag style="property: value"> </tag>
```

CSS Key Features:

- CSS specifies how HTML elements should be displayed on screens.
- The Major key feature of CSS includes styling rules which are interpreted by the client browser and applied to various elements.

- Using CSS, you can control the color text, the style of fonts, spacing between elements, how columns are sized, variation in display for different devices and screen size as well as a variety of other effects.

Selectors

To apply CSS to an element you need to select it. CSS provides you with a number of different ways to do this

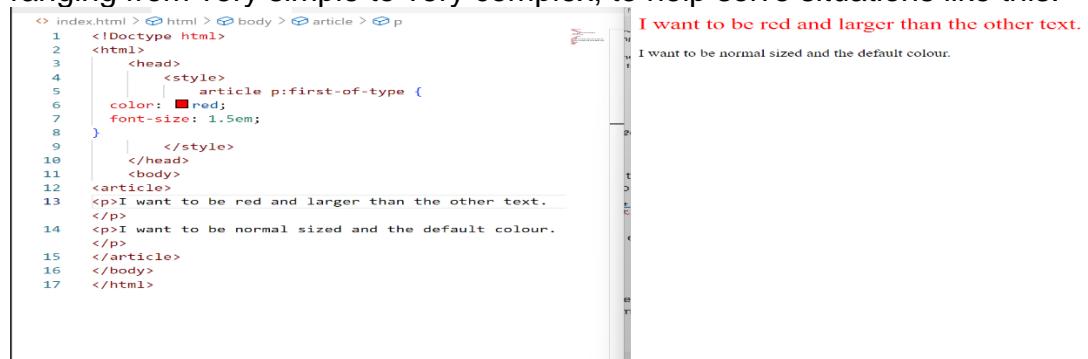
If you've got some text that you only want to be larger and red if it's the first paragraph of an article, how do you do that?

```
<article>
  <p>I want to be red and larger than the other text.</p>
  <p>I want to be normal sized and the default colour.</p>
</article>
```

You use a CSS selector to find that specific element and apply a CSS rule, like this.

```
article p:first-of-type {
  color: red;
  font-size: 1.5em;
}
```

CSS provides you with a lot of options to select elements and apply rules to them, ranging from very simple to very complex, to help solve situations like this.



```
<> index.html > html > body > article > p
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <style>
5        article p:first-of-type {
6          color: red;
7          font-size: 1.5em;
8        }
9      </style>
10     </head>
11     <body>
12       <article>
13         <p>I want to be red and larger than the other text.</p>
14         <p>I want to be normal sized and the default colour.</p>
15       </article>
16     </body>
17   </html>
```

The parts of a CSS rule

To understand how selectors work and their role in CSS, it's important to know the parts of a CSS rule. A CSS rule is a block of code, containing one or more selectors and one or more declarations.



In this CSS rule, the **selector** is `.my-css-rule` which finds all elements with a class of `my-css-rule` on the page. There are three declarations within the curly brackets. A declaration is a property and value pair which applies styles to the elements matched by the selectors.

Used to find or select the HTML elements you want to style. These are categorized as follows:

A **universal selector**—also known as a wildcard—matches any element.

```

* {
  color: hotpink;
}
  
```

This rule causes every HTML element on the page to have hotpink text.

Type selector

A **type selector** matches a HTML element directly.

```

section {
  padding: 2em;
}
  
```

This rule causes every `<section>` element to have 2em of padding on all sides.

Class selector

A HTML element can have one or more items defined in their class attribute. The class selector matches any element that has that class applied to it.

```

<div class="my-class"></div>
<button class="my-class"></button>
<p class="my-class"></p>
  
```

Any element that has the class applied to it will get coloured red:

```

.my-class {
  color: red;
}
  
```

A HTML element that has a class of .my-class will still be matched to the above CSS rule, even if they have several other classes, like this:

```
<div class="my-class another-class some-other-class"></div>
```

This is because CSS looks for a class attribute that *contains* the defined class, rather than matching that class exactly.

ID selector

An HTML element with an id attribute should be the only element on a page with that ID value. You select elements with an ID selector like this:

```
#rad {  
    border: 1px solid blue;  
}
```

This CSS would apply a blue border to the HTML element that has an id of rad, like this:

```
<div id="rad"></div>
```

Attribute selector

You can look for elements that have a certain HTML attribute, or have a certain value for an HTML attribute, using the attribute selector. Instruct CSS to look for attributes by wrapping the selector with square brackets ([]).

```
[data-type='primary'] {  
    color: red;  
}
```

This CSS looks for all elements that have an attribute of data-type with a value of primary, like this:

```
<div data-type="primary"></div>
```

Instead of looking for a specific value of data-type, you can also look for elements with the attribute present, regardless of its value.

```
[data-type] {  
    color: red;  
}  
<div data-type="primary"></div>  
<div data-type="secondary"></div>
```

Both of these <div> elements will have red text.

You can use case-sensitive attribute selectors by adding an s operator to your attribute selector.

```
[data-type='primary' s] {  
    color: red;  
}
```

This means that if a HTML element had a data-type of Primary, instead of primary, it would not get red text. You can do the opposite—case insensitivity—by using an i operator.

Along with case operators, you have access to operators that match portions of strings inside attribute values.

```
/* A href that contains "example.com" */
```

```
[href*='example.com'] {  
    color: red;  
}
```

```
/* A href that starts with https */
```

```
[href^='https'] {  
    color: green;  
}
```

```
/* A href that ends with .com */
```

```
[href$='.com'] {  
    color: blue;  
}
```

Grouping selectors

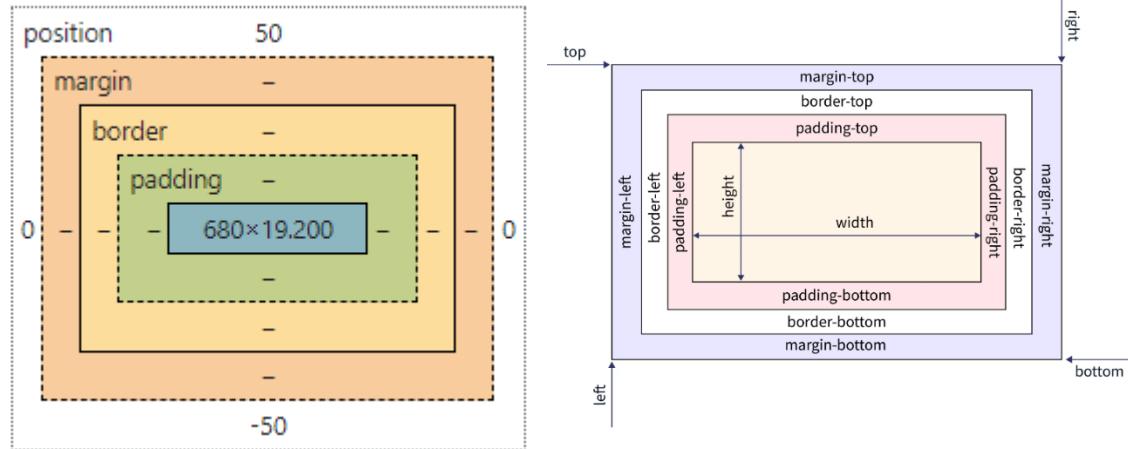
A selector doesn't have to match only a single element. You can group multiple selectors by separating them with commas:

```
strong,  
em,  
.my-class,  
[lang] {  
    color: red;  
}
```

What Is The CSS Box Model?

The CSS Box Model describes all the HTML elements of the webpage as rectangular boxes. As shown in the graphic below, let it be the logo, contents of the navigation bar or buttons, everything is a box.

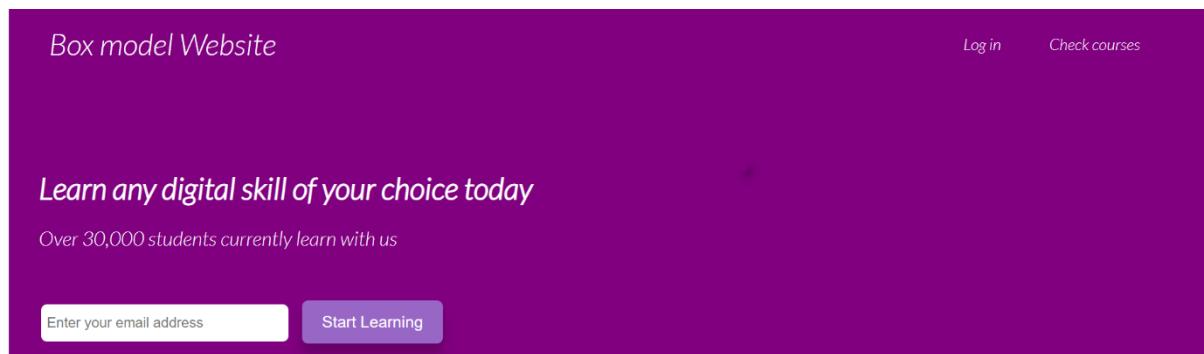
Diagram of The CSS Box Model



The CSS box model comprises the **box-sizing**, **padding** and **margin** properties.

If you **don't** use them, your website will look like this

But if you use the box model properties correctly, your website will look like this



How to Setup the Project

HTML

Open VS Code or Notepad++ and write this code inside the body tag:

```
<div class="box-1"> Box-1 </div>
```

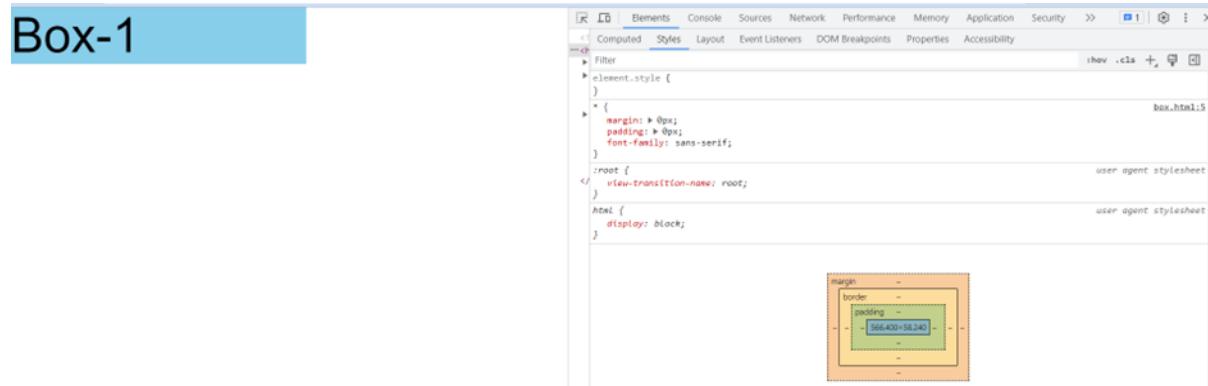
CSS

Clear the default styles of our browser

```
* {  
    margin: 0px;  
    padding: 0px;  
    font-family: sans-serif;  
}
```

Now, let's style our box

```
.box-1 {  
    width: 300px;  
    background-color: skyblue;  
    font-size: 50px;  
}
```



The Padding Property

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

How to use the padding property in CSS

This is the shorthand of the four padding properties:

- padding-top
- padding-right
- padding-bottom
- padding-left



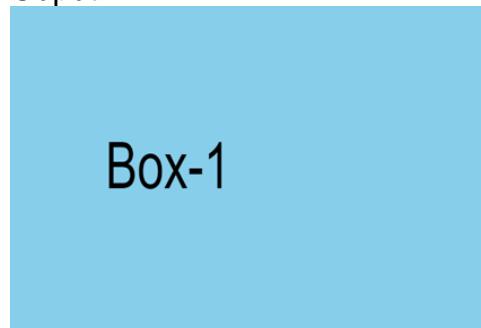
Let's add some padding to our content.

// Padding added on top, right, left, bottom of .box-1

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      * {
        margin: 0px;
        padding: 0px;
        font-family: sans-serif;
      }
      .box-1 {
        /*Padding added on top, right, left, bottom of .box-1*/
        padding: 100px;

        width: 300px;
        background-color: skyblue;
        font-size: 50px;
      }
    </style>
  </head>
  <body>
```

Output:



A screenshot of the Chrome DevTools Elements tab. It shows the CSS rules for the `.box-1` class and the user agent stylesheet. The box model is visualized as a nested set of rectangles: an outermost orange dashed border representing the margin, followed by a green dashed border for the border, a blue solid border for the padding, and a central light blue area for the content. The dimensions shown are 300px width and 50px font size, resulting in a total height of 350px.

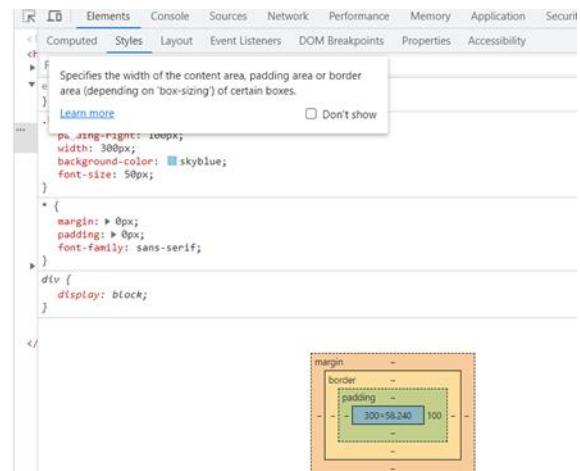
Let's try adding padding to just 1 side of our content (only the right side):

```
.box-1{
  padding: 0 100px 0 0;
```

// Or you can use

```
.box-1{
```

```
padding-right: 100px;  
}
```



The Border Property

The CSS border properties allow you to specify the style, width, and color of an element's border.

How to use the border property in CSS

the **border** is the space added on top of our **main content + padding**

There are three crucial inputs of the border property:

- border size : in px
 - border style : solid / dotted/ dashed
 - border color : name of color

border

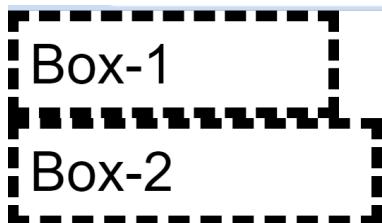
```
.box-1 {  
    width: 300px;  
    font-size: 50px;  
    padding: 50px;  
    border: 10px dashed black;  
}
```

```

<style>
  * {
    margin: 0px;
    padding: 0px;
    font-family: sans-serif;
  }
  .box-1 {
    box-sizing: border-box;
    width: 300px;
    font-size: 50px;
    padding: 10px;
    border: 10px dashed black;
  }
  .box-2 {
    box-sizing: content-box;
    width: 300px;
    font-size: 50px;
    padding: 10px;
    border: 10px dashed black;
  }
</style>

```

Output:



The diagram shows a central green square representing the content area. It is surrounded by a white border, which is further surrounded by a larger orange border. The total width of the element is 300px, and the font size is 50px. The padding is 10px, and the border is 10px. The margins are 9.6px on all four sides (top, bottom, left, right).

The Margin Property

The CSS margin properties are used to create space around elements, outside of any defined borders.

How to use margin property in CSS

This is the shorthand for the four properties of the margin property:

- margin-top
- margin-right
- margin-bottom
- margin-left

margin : 10px 20px 15px 12px
 Top bottom
 right left

```
.box-1 {
  padding: 50px;
  border: 10px dashed black;
  margin: 50px;
}
```

Output:

The screenshot shows the 'Computed' tab in the developer tools. It displays the following CSS rules:

```
element.style {
  font-family: sans-serif;
}
.box-1 {
  padding: 50px;
  border: 10px dashed black;
  margin: 50px;
}
div {
  display: block;
}
```

A box model diagram on the right illustrates the layout with an orange background. The total width and height are 50px each. The outermost layer is labeled 'margin' with a value of 50. Inside is a black border with a value of 9.600. Then there's a white area labeled 'padding' with a value of 50. The innermost content area is labeled 'border' with a value of 9.600.

Let's try adding a margin to just 1 side of our content (only the left side):

```
<style>
  * {
    margin: 0px;
    padding: 0px;
    font-family: sans-serif;
  }
  .box-1 {
    padding: 50px;
    border: 10px dashed black;
    margin-left: 50px;
  }
</style>
</head>
<body>
  <div class="box-1"> Box-1 </div>
</body>
</html>
```

Output:

The screenshot shows the 'Computed' tab in the developer tools. It displays the following CSS rules:

```
element.style {
  font-family: sans-serif;
}
div {
  display: block;
}
.box-1 {
  padding: 50px;
  border: 10px dashed black;
  margin-left: 50px;
}
```

A box model diagram on the right illustrates the layout. The total width is 50px. The left side has a margin of 50 and a border of 9.600. The right side has a margin of 9.600 and a border of 9.600. The top and bottom sides have margins of 9.600 and borders of 9.600. The inner content area has a padding of 50.

CSS Colors/Backgrounds

Using Color Name

Example Code:

```

<html>
  <head>
    <style>
      :root {
        --main-bg-color: cyan;
        --main-text-color: pink;
      }

      body {
        background-color: var(--main-bg-color);
        color: var(--main-text-color);
        text-align: center;
      }

    </style>
  </head>
  <body>
    <h1>CSS Color</h1>
  </body>
</html>

```

Output:



Using HEX Format

Hexadecimal can be defined as a six-digit color representation. This notation starts with the # symbol followed by six characters ranges from 0 to F. In hexadecimal notation, the first two digits represent the red (RR) color value, the next two digits represent the green (GG) color value, and the last two digits represent the blue (BB) color value.

The black color notation in hexadecimal is #000000, and the white color notation in hexadecimal is #FFFFFF. Some of the codes in hexadecimal notation are #FF0000, #00FF00, #0000FF, #FFFF00, and many more.

color:#(0-F)(0-F)(0-F)(0-F)(0-F)(0-F);

Example Code:

```

<html>
  <head>
    <style>
      :root {
        --main-bg-color: #aeaeae;
        --main-text-color: #000;
      }

      body {
        background-color: var(--main-bg-color);
        color: var(--main-text-color);
        text-align: center;
      }
    </style>
  </head>
  <body>
    <h1>CSS Color</h1>
  </body>
</html>

```

Output:



Using RGB Format

RGB format is the short form of 'RED GREEN and BLUE' that is used for defining the color of an HTML element simply by specifying the values of R, G, B that are in the range of 0 to 255.

The color values in this format are specified by using the `rgb()` property. This property allows three values that can either be in percentage or integer (range from 0 to 255). This property is not supported in all browsers; that's why it is not recommended to use it.

Syntax

`color: rgb(R, G, B);`

Example Code:

```

<html>
  <head>
    <style>
      :root {
        --main-bg-color: #rgb(238,130,238);
        --main-text-color: #rgb(255,255,255);
      }

      body {
        background-color: var(--main-bg-color);
        color: var(--main-text-color);
        text-align: center;
      }

    </style>
  </head>
  <body>
    <h1>CSS Color</h1>

  </body>
</html>

```

Output:



Using HSL Format

It is a short form of Hue, Saturation, and Lightness. Let's understand them individually.

Hue: It can be defined as the degree on the color wheel from 0 to 360. 0 represents red, 120 represents green, 240 represents blue.

Saturation: It takes value in percentage in which 100% represents fully saturated, i.e., no shades of gray, 50% represent 50% gray, but the color is still visible, and 0% represents fully unsaturated, i.e., completely gray, and the color is invisible.

Lightness: The lightness of the color can be defined as the light that we want to provide the color in which 0% represents black (there is no light), 50% represents neither dark nor light, and 100% represents white (full lightness).

Example Code:

```
<html>
  <head>
    <style>
      :root {
        --main-bg-color: hsl(0,50%,50%);
        --main-text-color: hsla(0,50%,0,0.5);
      }

      body {
        background-color: var(--main-bg-color);
        color: var(--main-text-color);
        text-align: center;
      }

    </style>
  </head>
  <body>
    <h1>CSS Color</h1>

  </body>
</html>
```

Output:



CSS Text Property

Text-Align

The text-align-last property in CSS is used to set the last line of the paragraph just before the line break.

Syntax:

text-align-last: auto | start | end | left | right | center | justify | initial | inherit;

Default Value : Its default value is auto.

Example Code

```

<style>
.a {
    text-align-last: right;
    font-family: sans-serif;
    border: 1px solid black;
}
.b {
    text-align-last: left;
    font-family: sans-serif;
    border: 1px solid black;
}
.c {
    text-align-last: center;
    font-family: sans-serif;
    border: 1px solid black;
}
.d {
    text-align-last: justify;
    font-family: sans-serif;
    border: 1px solid black;
}

```

Output:

Prepare for the Recruitment drive of product based companies like Microsoft,Amazon, Adobe etc with a free online placement preparation course. The course focuses on various MCQ's & Coding question likely to be asked in the interviews & make your upcoming placement season efficient and successful.

Prepare for the Recruitment drive of product based companies like Microsoft,Amazon, Adobe etc with a free online placement preparation course. The course focuses on various MCQ's & Coding question likely to be asked in the interviews & make your upcoming placement season efficient and successful.

Prepare for the Recruitment drive of product based companies like Microsoft,Amazon, Adobe etc with a free online placement preparation course. The course focuses on various MCQ's & Coding question likely to be asked in the interviews & make your upcoming placement season efficient and successful.

Prepare for the Recruitment drive of product based companies like Microsoft,Amazon, Adobe etc with a free online placement preparation course. The course focuses on various MCQ's & Coding question likely to be asked in the interviews & make your upcoming placement season efficient and successful.

Text-Overflow Property

The text-overflow property specifies how overflowed content that is not displayed should be signaled to the user. It can be clipped, display an ellipsis (...), or display a custom string.

Both of the following properties are required for text-overflow:

- white-space: nowrap;
- overflow: hidden;

Columns Property

The columns property is a shorthand property for:

- column-width
- column-count

The column-width part will define the minimum width for each column, while the column-count part will define the maximum number of columns

Example Code

```

<!DOCTYPE html>
<html>
<head>
<style>
.a {
| columns: 100px 3;
}
</style>
</head>
<body>
<h1>The columns Property</h1>
<p>The columns property is a shorthand property for column-width and column-count:</p>
<p><strong>Set the minimum width for each column to 100px, and the maximum number of columns to 3:</strong></p>
<div class="a">
Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.
</div>
</body>
</html>

```



Output:

The columns Property

The columns property is a shorthand property for column-width and column-count:

Set the minimum width for each column to 100px, and the maximum number of columns to 3:

 Lorem ipsum dolor sit amet, consecetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure

 dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option

 congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

Text-indent Property

The text-indent property in CSS is used to define the indentation of the first line in each block of text. It also takes negative values. It means if the value is negative then the first line will be indented to the left.

Syntax:

text-indent: length | initial | inherit;

Property values

- **length:** It is used to set fixed indentation in terms of px, pt, cm, em etc. The default value of length is 0.

Text-shadow Property

The text-shadow property in CSS is used to add shadows to the text. This property accepts a list of shadows that are to be applied to the text, separated by the comma. The default value of the text-shadow property is none.

Syntax:

text-shadow: h-shadow v-shadow blur-radius color | none | initial | inherit;

Property values:

- **h-shadow:** This property is required & used to specify the position of horizontal shadow. It accepts the negative values.
- **v-shadow:** This property is required & used to specify the position of vertical shadow. It also accepts the negative values.
- **blur-radius:** It is used to set the blur radius. Its default value is 0 & is optional.

- none: It represents no shadow added to the text, this is the default value.
- color: It is used to set the color of the shadow. It is optional.
- initial: It is used to set text-shadow to its default value.
- inherit: This property is inherited from its parent element.

Example Code:

```
<!DOCTYPE html>
<html>
<head>
    <title> CSS | text-shadow Property </title>
    <style>
        h1 {
            text-shadow: 5px 5px 8px #00FF00;
        }
    </style>
</head>

<body>
    <h1> TEXT SHADOW </h1>
</body>
</html>
```

Output:



Line height Property

The line-height property in CSS is used to set the amount of space used for lines, such as in the text. Negative values are not allowed.

Syntax:

line-height: normal | number | length | percentage | initial | inherit;

Property values:

- normal: This mode represents the normal line height. This is the default value.
- initial: This mode is used to set this property to its default value.
- number: This value is a unitless number multiplied by the current font-size to set the line height. In most cases, this is the preferred way to set line height and avoid unexpected results due to inheritance.
- length: In this mode a fixed line height is specified.

- percentage: This mode is used to set line height in percent of the current font size.

Example Code

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS line-height Property</title>
    <style>
        .a {
            line-height: 2.5;
            background: #pink;
            color: #white;
        }
        .b {
            line-height: 2em;
            background: #lightblue;
            color: #white;
        }
        .c {
            line-height: 150%;
            background: #lightseagreen;
            color: #white;
        }
    </style>
</head>

<body>
    <div class="a">
        This div has line-height: 2em;
    </div>
    <div class="b">
        This div has line-height: 2em;
    </div>
    <div class="c">
        This div has line-height: 2em;
    </div>
</body>
```

Output:

A computer science portal for geeks.

This div has line-height: 2em;

A computer science portal for geeks.

This div has line-height: 2em;

A computer science portal for geeks.

This div has line-height: 2em;

CSS FONTS

Font Family

Using a font that is easy to read is important. The font adds value to your text. It is also important to choose the correct color and text size for the font.

Generic Font Families

In CSS there are five generic font families:

1. Serif fonts have a small stroke at the edges of each letter. They create a sense of formality and elegance.

2. Sans-serif fonts have clean lines (no small strokes attached). They create a modern and minimalistic look.
3. Monospace fonts - here all the letters have the same fixed width. They create a mechanical look.
4. Cursive fonts imitate human handwriting.
5. Fantasy fonts are decorative/playful fonts.

we use the font-family property to specify the font of a text.

Example Code

```
<!DOCTYPE html>
<html>
<head>
<style>
.p1 {
| font-family: "Times New Roman", Times, serif;
}

.p2 {
| font-family: Arial, Helvetica, sans-serif;
}

.p3 {
| font-family: "Lucida Console", "Courier New", monospace;
}
</style>
</head>
<body>

<h1>CSS font-family</h1>
<p class="p1">This is a paragraph, shown in the Times New Roman font.</p>
<p class="p2">This is a paragraph, shown in the Arial font.</p>
<p class="p3">This is a paragraph, shown in the Lucida Console font.</p>
</body>
</html>
```

Output:

CSS font-family

This is a paragraph, shown in the Times New Roman font.

This is a paragraph, shown in the Arial font.

This is a paragraph, shown in the Lucida Console font.

Font Style

The font-style property is mostly used to specify italic text. The font-weight property specifies the weight of a font. The font-variant property specifies whether or not a text should be displayed in a small-caps font.

Font Size

The font-size property sets the size of the text. Setting the text size with pixels, em, vw, and percentage gives you full control over the text size.

CSS Google Fonts

If you do not want to use any of the standard fonts in HTML, you can use Google Fonts. Google Fonts are free to use, and have more than 1000 fonts to choose from.

Example Code

```
<!DOCTYPE html>
<html>
<head>
|   <link href="https://fonts.googleapis.com/css2?family=Monofett&display=swap" rel="stylesheet">
<style>
h1{font-family: 'Monofett', monospace;}
</style>
</head>
<body>

<h1>This is heading 1</h1>

</body>
</html>
```

Output:



CSS Position

The position property in CSS tells about the method of positioning for an element or an HTML entity and the positioning of an element can be done using the top, right, bottom, and left properties. These specify the distance of an HTML element from the edge of the viewport.

There are five different types of position properties available in CSS:

- **Fixed:** Any HTML element with position: fixed property will be positioned relative to the viewport. An element with fixed positioning allows it to remain in the same position even we scroll the page. We can set the position of the element using the top, right, bottom, and left.
- **Static:** This method of positioning is set by default. If we don't mention the method of positioning for any element, the element has the position: static method by default. By defining Static, the top, right, bottom, and left will not have any control over the element. The element will be positioned with the normal flow of the page.
- **Relative:** An element with position: relative is positioned relatively with the other elements which are sitting on top of it. If we set its top, right, bottom, or left, other elements will not fill up the gap left by this element. An element with its position set to relative and when adjusted using top, bottom, left, and right will be positioned relative to its original position.
- **Absolute:** An element with position: absolute will be positioned with respect to its nearest Non-static ancestor. The positioning of this element does not depend upon its siblings or the elements which are at the same level.
- **Sticky:** Element with position: sticky and top:0 played a role between fixed & relative based on the position where it is placed. If the element is placed in the middle of the document then when the user scrolls the document, the sticky element start scrolling until it touched the top. When it touches the top, it will

be fixed at the top place in spite of further scrolling. we can stick the element at the bottom, with the bottom property.

Example Code

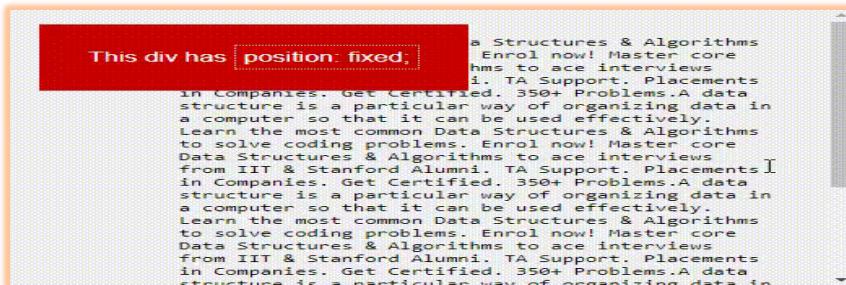
```
<!DOCTYPE html>
<html>
<head>
    <title> CSS Positioning Element</title>
    <style>
        body {
            margin: 0;
            padding: 20px;
            font-family: sans-serif;
            background: #eefefef;
        }

        .fixed {
            position: fixed;
            background: #cc0000;
            color: #ffffff;
            padding: 30px;
            top: 50px;
            left: 10px;
        }

        span {
            padding: 5px;
            border: 1px solid #ffff00 dotted;
        }
    </style>
</head>

<body>
    <div class="fixed">This div has
        <span>position: fixed;</span>
    </div>
    <pre>
        Learn the most common Data Structures & Algorithms
        to solve coding problems. Enrol now! Master core
        Data Structures & Algorithms to ace interviews
        from IIT & Stanford Alumni. TA Support. Placements
        in Companies. Get Certified. 350+ Problems.A data
        structure is a particular way of organizing data in
        a computer so that it can be used effectively.
        Learn the most common Data Structures & Algorithms
        to solve coding problems. Enrol now! Master core
        Data Structures & Algorithms to ace interviews
        from IIT & Stanford Alumni. TA Support. Placements
        in Companies. Get Certified. 350+ Problems.A data
        structure is a particular way of organizing data in
        a computer so that it can be used effectively.
        Learn the most common Data Structures & Algorithms
        to solve coding problems. Enrol now! Master core
        Data Structures & Algorithms to ace interviews
        from IIT & Stanford Alumni. TA Support. Placements
        in Companies. Get Certified. 350+ Problems.A data
        structure is a particular way of organizing data in
        a computer so that it can be used effectively.
        Learn the most common Data Structures & Algorithms
        to solve coding problems. Enrol now! Master core
        Data Structures & Algorithms to ace interviews
        from IIT & Stanford Alumni. TA Support. Placements
        in Companies. Get Certified. 350+ Problems.A data
        structure is a particular way of organizing data in
        a computer so that it can be used effectively.
    </pre>
</body>
</html>
```

Output for Fixed Position:



Output for Static Position:

This div has `position: static;`

Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively. Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively. Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively. Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively.

Output for Relative Position:

Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively. Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively.

This div has `position: relative;`

Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in

Output for Absolute Position:

Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively. Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core Data Structures & Algorithms to ace interviews from IIT & Stanford Alumni. TA Support. Placements in Companies. Get Certified. 350+ Problems. A data structure is a particular way of organizing data in a computer so that it can be used effectively.

This div has `position: relative;`

This div has `position: absolute;`

Learn the most common Data Structures & Algorithms to solve coding problems. Enrol now! Master core

Output for Sticky Position:



CSS List

The List in CSS specifies the listing of the contents or items in a particular manner i.e., it can either be organized orderly or unorder way, which helps to make a clean webpage. It can be used to arrange the huge with a variety of content as they are flexible and easy to manage. The default style for the list is borderless.

The list can be categorized into 2 types:

- Unordered List: In unordered lists, the list items are marked with bullets i.e. small black circles by default.
- Ordered List: In ordered lists, the list items are marked with numbers and an alphabet.

We have the following CSS lists properties, which can be used to control the CSS lists:

- **list-style-type:** This property is used to specify the appearance (such as disc, character, or custom counter style) of the list item marker.
- **list-style-image:** This property is used to set the images that will be used as the list item marker.
- **list-style-position:** It specifies the position of the marker box with respect to the principal block box.
- **list-style:** This property is used to set the list style.

CSS Table

`<table>` element was the only possible avenue for creating rich design layouts on the Web. But creating layout with `<table>` was not its intended or ideal use. Now that better layout options are available, developers can use the `<table>` element for presenting tabular data as intended, much like a spreadsheet. This allows for semantic HTML, or using HTML elements in alignment with their intended meaning.

Pseudo-elements

A pseudo-element is like adding or targeting an extra element without having to add more HTML.

Syntax:

```
selector::pseudo-element {  
    property: value;  
}
```

There are many Pseudo Elements in CSS but the ones which are most commonly used are as follows:

- **::first-line:** ::first-line Pseudo-element applies styles to the first line of a block-level element. Note that the length of the first line depends on many factors, including the width of the element, the width of the document, and the font size of the text. Note that only a few properties are applied for first-line pseudo-element like font properties, color properties, background properties, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, clear, etc.
- **::first-letter:** ::first-letter Pseudo-element applies styles to the first letter of the first line of a block-level element, but only when not preceded by other content (such as images or inline tables). Note that only a few properties are applied for first-line pseudo-element like font properties, color properties, background properties, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, clear, etc.
- **::before:** ::before Pseudo-element creates a pseudo-element that is the first child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.
- **::after:** ::after Pseudo-element creates a pseudo-element that is the last child of the selected element. It is often used to add cosmetic content to an element with the content property. It is inline by default.

Example Code

```
<title>first-line Demo</title>  
<style>  
    body{  
        background-color: #whitesmoke;  
        color: #green;  
        font-size: large;  
        text-align: center;  
    }  
.a::first-line{  
    color: #Red;  
    font-weight: bold;  
}  
.b::first-letter{  
    color: #Red;  
    font-size: 30px;  
    font-weight: bold;  
}  
.c::before{  
    content: "";  
    color: #red;  
    font-size: 30px;  
}  
.d::after{  
    content: "";  
    color: #red;  
    font-size: 30px;  
}  
</style>  
</head>
```

```

<body>
  <h2>::first-line element</h2>
  <p class="a">This is a paragraph using first-line pseudo-element
  to style first line of the paragraph.Content in the
  first line turns red and becomes bold.</p>
  <p class="b">This is a paragraph using first-line pseudo-element
  to style first line of the paragraph.Content in the
  first line turns red and becomes bold.</p>
  <p class="c">This is a paragraph using first-line pseudo-element
  to style first line of the paragraph.Content in the
  first line turns red and becomes bold.</p>
  <p class="d">This is a paragraph using first-line pseudo-element
  to style first line of the paragraph.Content in the
  first line turns red and becomes bold.</p>
</body>
</html>

```

Output:

::first-line element

This is a paragraph using first-line pseudo-element to style first line of the paragraph.Content in the first line turns red and becomes bold.

This is a paragraph using first-line pseudo-element to style first line of the paragraph.Content in the first line turns red and becomes bold.

" This is a paragraph using first-line pseudo-element to style first line of the paragraph.Content in the first line turns red and becomes bold.

This is a paragraph using first-line pseudo-element to style first line of the paragraph.Content in the first line turns red and becomes bold."

CSS Transformation

The transform property in CSS is used to change the coordinate space of the visual formatting model. This is used to add effects like skew, rotate, translate, etc on elements.

Syntax:

transform: none | transform-functions | initial | inherit;

Example Code

```

<body>
  <div class="main">
    <div class="a">CSS TRANSFORM</div>
  </div>
  <div class="main">
    <div class="b">CSS TRANSFORM</div>
  </div>
  <div class="main">
    <div class="c">CSS TRANSFORM</div>
  </div>
  <div class="main">
    <div class="d">CSS TRANSFORM</div>
  </div>
  <div class="main">
    <div class="e">CSS TRANSFORM</div>
  </div>
</body>

```

```

<style>
.main {
    display: grid;
    padding: 30px;
    background-color: #green;
}

.a {
    text-align: center;
    font-size: 35px;
    background-color: #white;
    color: #green;
    transform: matrix(1, 0, -1, 1, 1, 0);
}

.b {
    text-align: center;
    font-size: 35px;
    background-color: #white;
    color: #green;
    transform-style: preserve-3d;
    position: absolute;
    transform: translate(150px, 75%, 5em)
}

.c {
    text-align: center;
    font-size: 35px;
    background-color: #white;
    color: #green;
    transform: rotate(45deg);
}

.d {
    text-align: center;
    font-size: 35px;
    background-color: #white;
    color: #green;

    transform: scale(1, 2);
}

.e {
    text-align: center;
    font-size: 35px;
    background-color: #white;
    color: #green;
    transform: skew(30deg, 30deg);
}

```

Output:



CSS Animations

CSS Animations is a technique to change the appearance and behavior of various elements in web pages. It is used to control the elements by changing their motions or display. It has two parts, one contains the CSS properties which describe the animation of the elements and the other contains certain keyframes which indicate the animation properties of the element and the specific time intervals at which those have to occur.

What is a Keyframe?

Keyframes are the foundations with the help of which CSS Animations work. They define the display of the animation at the respective stages of its whole duration.

- **animation-name:** It is used to specify the name of the @keyframes describing the animation.

- **animation-duration:** It is used to specify the time duration it takes animation to complete one cycle.
- **animation-timing-function:** It specifies how animations make transitions through keyframes. There are several presets available in CSS which are used as the value for the animation-timing-function like linear, ease, ease-in, ease-out, and ease-in-out.
- **animation-delay:** It specifies the delay of the start of an animation.
- **animation-iteration-count:** This specifies the number of times the animation will be repeated.
- **animation-direction:** It defines the direction of the animation. animation direction can be normal, reverse, alternate, and alternate-reverse.
- **animation-fill-mode:** It defines how styles are applied before and after animation. The animation fill mode can be none, forwards, backwards, or both.
- **animation-play-state:** This property specifies whether the animation is running or paused.

Example Code:

```

<style>
    #main {
        animation-name: color;
        animation-duration: 25s;
        padding-top: 30px;
        padding-bottom: 30px;
        font-family: Times New Roman;
    }

    #a {
        font-size: 40px;
        text-align: center;
        font-weight: bold;
        color: #090;
        padding-bottom: 5px;
    }

    #b {
        font-size: 17px;
        font-weight: bold;
        text-align: center;
    }

    @keyframes color {
        0% {
            background-color: red;
        }

        50% {
            background-color: orange;
        }

        100% {
            background-color: brown;
        }
    }
</style>
</head>

<body>
    <div id="main">
        <div id="a">EDUNET FOUNDATION</div>
        <div id="b">TECHSAKSHAM</div>
    </div>
</body>

```

Output:

CSS Media Queries

The Media query in CSS is used to create a responsive web design. It means that the view of a web page differs from system to system based on screen or media types. The breakpoint specifies for what device-width size, the content is just starting to break or deform.

Media queries can be used to check many things:

- width and height of the viewport
- width and height of the device
- Orientation
- Resolution

A media query consist of a media type that can contain one or more expression which can be either true or false. The result of the query is true if the specified media matches the type of device the document is displayed on. If the media query is true then a style sheet is applied.

Syntax:

```
@media not | only mediatype and (expression) {  
    // Code content  
}
```

Laptop View:

```
<!DOCTYPE html>  
<html>  
<head>  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<style>  
body {  
    background-color: lightgreen;  
}  
  
@media only screen and (max-width: 600px) {  
    body {  
        background-color: lightblue;  
    }  
}</style>  
</head>  
<body>
```

Resize the browser window. When the width of this document is 600 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".



Mobile View:

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport"
      content="width=device-width, initial-
      scale=1.0">
<style>
body {
    background-color: lightgreen;
}

@media only screen and (max-width:
600px) {
    body {
        background-color: lightblue;
    }
}
</style>
</head>
<body>

<p>Resize the browser window. When the width of this
document is 600 pixels or less, the background-color is
"lightblue", otherwise it is
"lightgreen".</p>

</body>
</html>

```

Resize the browser window. When the width of this document is 600 pixels or less, the background-color is "lightblue", otherwise it is "lightgreen".

CSS Grid

CSS Grid is a powerful tool that allows for two-dimensional layouts to be created on the web.

The `display: grid` directive creates only a single-column grid container. Therefore, the grid items will display in the normal layout flow (one item below another). `inline-grid` tells browsers to display the selected HTML element as an inline-level grid box model. In other words, setting an element's `display` property's value to `inline-grid` turns the box model into an inline-level grid layout module.

Grid Container

Create a grid container by setting the `display` property with a value of `grid` or `inline-grid`. All direct children of grid containers become grid items.

Example

`display: grid`

Grid items are placed in rows by default and span the full width of the grid container.



`display: inline-grid`



Explicit Grid

Explicitly set a grid by creating columns and rows with the `grid-template-columns` and `grid-template-rows` properties.

grid-template-rows: 50px 100px

A row track is created for each value specified for grid-template rows. Track size values can be any non-negative, length value (px, %, em, etc.). Items 1 and 2 have fixed heights of 50px and 100px. Because only 2-row tracks were defined, the heights of items 3 and 4 are defined by the contents of each.

1	2	3
4	5	6

grid-template-columns: 90px 50px 120px

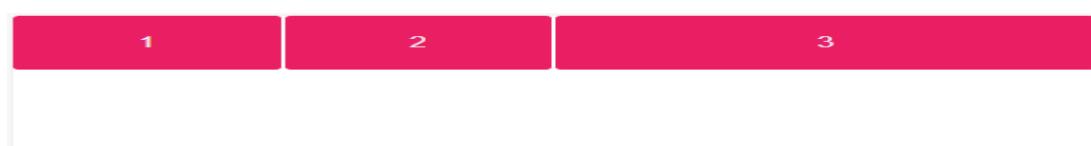
Like rows, a column track is created for each value specified for grid-template-columns. Items 4, 5 and 6 were placed on a new row track because only 3 column track sizes were defined; and because they were placed in column tracks 1, 2 and 3, their column sizes are equal to items 1, 2 and 3. Grid items 1, 2 and 3 have fixed widths of 90px, 50px and 120px respectively



grid-template-columns: 1fr 1fr 2fr

The fr unit helps create flexible grid tracks. It represents a fraction of the available space in the grid container (works like Flexbox's unitless values).

In this example, items 1 and 2 take up the first two (of four) sections while item 3 takes up the last two.

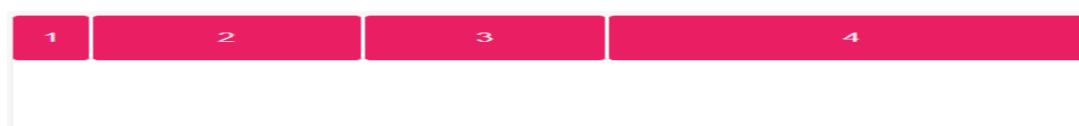


grid-template-columns: 3rem 25% 1fr 2fr

fr is calculated based on the remaining space when combined with other length values.

In this example, 3rem and 25% would be subtracted from the available space before the size of fr is calculated:

$$1\text{fr} = ((\text{width of grid}) - (3\text{rem}) - (25\% \text{ of width of grid})) / 3$$



Define repeating grid tracks using the repeat() notation. This is useful for grids with items with equal sizes or many items.

```
grid-template-rows: repeat(4, 100px);  
grid-template-columns: repeat(3, 1fr);
```

The repeat() notation accepts 2 arguments: the first represents the number of times the defined tracks should repeat, and the second is the track definition.

1	2	3
4	5	6
7	8	9
10	11	12

grid-template-columns: 30px repeat(3, 1fr) 30px

repeat() can also be used within track listings. In this example, the first and last column tracks have widths of 30px, and the 3 column tracks in between, created by repeat(), have widths of 1fr each.

1	2	3	4	5
6	7	8	9	10

Grid Gaps (Gutters)

The grid-column-gap and grid-row-gap properties create gutters between columns and rows. Grid gaps are only created in between columns and rows, and not along the edge of the grid container

```
grid-row-gap: 20px;  
grid-column-gap: 5rem;
```

Gap size values can be any non-negative, length value (px, %, em, etc.)

1	2
3	4

grid-gap: 100px 1em

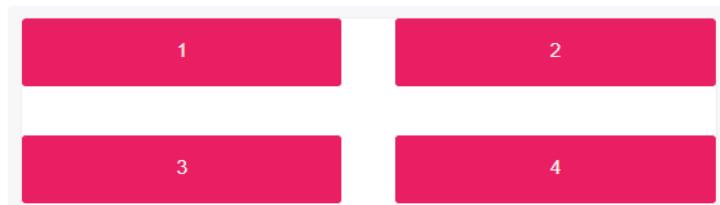
grid-gap is shorthand for grid-row-gap and grid-column-gap.

If two values are specified, the first represents grid-row-gap and the second grid-column-gap.



grid-gap: 2rem

One value sets equal row and column gaps.

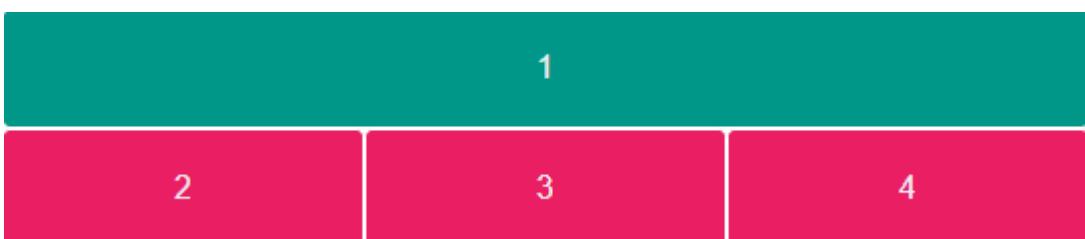


Spanning Items Across Rows and Columns

grid-column-start: 1;

grid-column-end: 4;

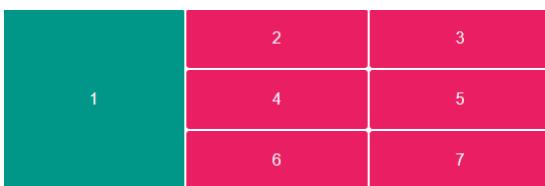
Set a grid item to span more than one column track by setting grid-column-end to a column line number that is more than one column away from grid-column-start.



grid-row-start: 1;

grid-row-end: 4;

Grid items can also span across multiple row tracks by setting grid-row-end to more than one row track away.



Like grid line names, grid areas can also be named with the grid-template-areas property. Names can then be referenced to position grid items.

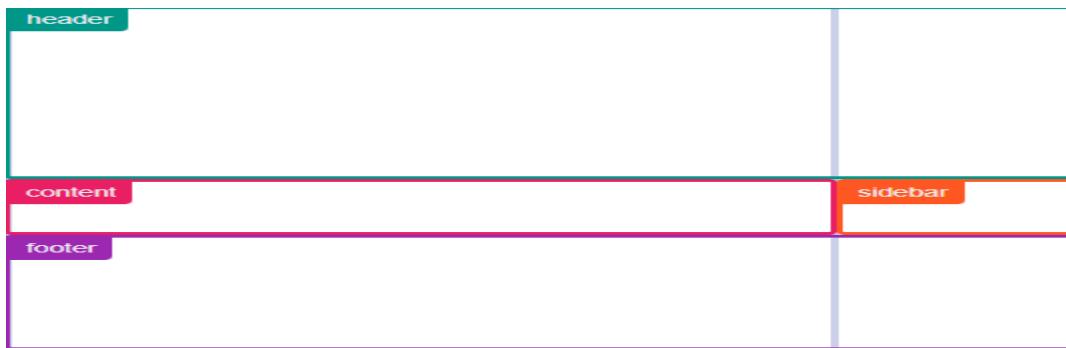
grid-template-areas: "header header"
"content sidebar"

```

    "footer footer";
grid-template-rows: 150px 1fr 100px;
grid-template-columns: 1fr 200px;

```

Sets of names should be surrounded in single or double quotes, and each name separated by a whitespace. Each set of names defines a row, and each name defines a column.



Aligning Grid Items (Box Alignment)

CSS's Box Alignment Module complements CSS Grid to allow items to be aligned along the row or column axis. `justify-items` and `justify-self` align items along the row axis, and `align-items` and `align-self` align items along the column axis.

`justify-items` and `align-items` are applied to the grid container and support the following values:

auto, normal, start, end, center, stretch, baseline

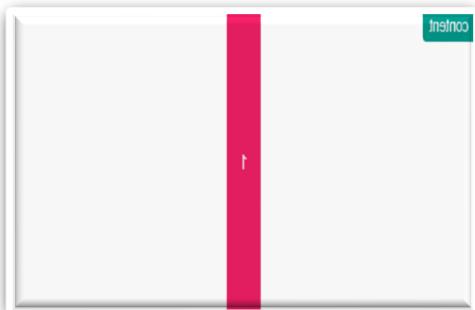
```

.grid {
  grid-template-rows: 80px 80px;
  grid-template-columns: 1fr 1fr;
  grid-template-areas: "content content"
                      "content content";
}
.item { grid-area: content }
.grid {
  justify-items: start
}

```



`justify-items: center`

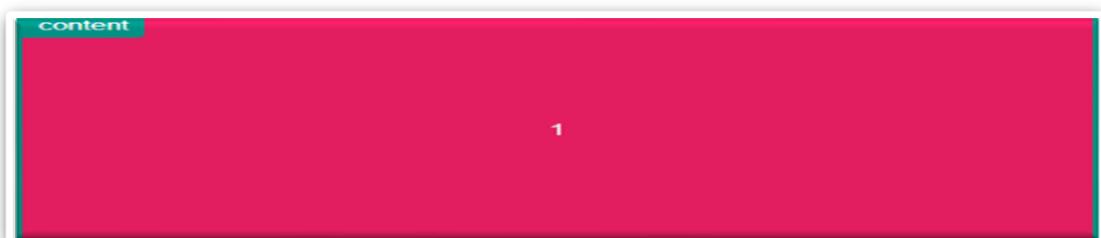


justify-items: end



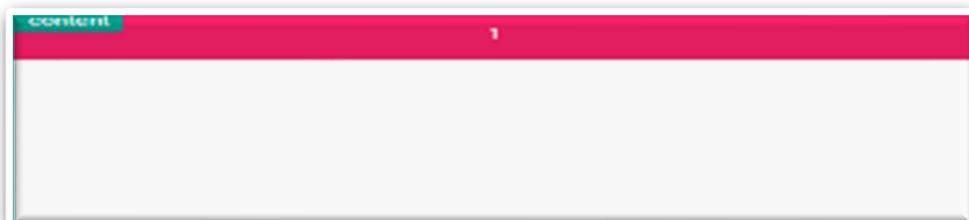
justify-items: stretch

Items are stretched across the entire row axis. stretch is the default value.



align-items: start

Items are positioned at the start of the column axis (column line 1).



justify-items: center

align-items: center

Items are positioned at the center of the row and column axes.



CSS FLEX

CSS3 Flexible boxes also known as CSS Flexbox, is a new layout mode in CSS3. The CSS3 flexbox is used to make the elements behave predictably when they are used with different screen sizes and different display devices. It provides a more efficient way to layout, align and distribute space among items in the container. It is mainly used to make CSS3 capable to change its item's width and height to best fit all available spaces. It is preferred over the block model.

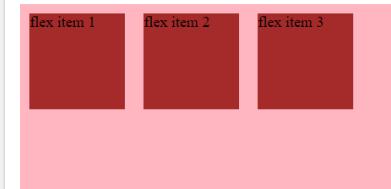
The CSS3 flexbox contains flex containers and flex items.

- **Flex container:** The flex container specifies the properties of the parent. It is declared by setting the display property of an element to either flex or inline-flex.
- **Flex items:** The flex items specify properties of the children. There may be one or more flex items inside a flex container.



Example Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 400px;
    height: 200px;
    background-color: lightpink;
}
.flex-item {
    background-color: brown;
    width: 100px;
    height: 100px;
    margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
    <div class="flex-item">flex item 1</div>
    <div class="flex-item">flex item 2</div>
    <div class="flex-item">flex item 3</div>
</div>
</body>
</html>
```



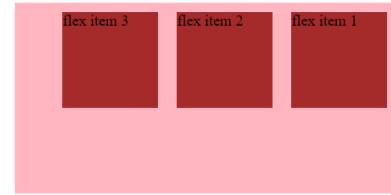
Flex Direction Property

The flex-direction property is used to set the direction of the flexible items inside the flex container. Its default value is row (left-to-right, top-to-bottom).

```

<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-direction: row-reverse;
  width: 400px;
  height: 200px;
  background-color: lightpink;
}
.flex-item {
  background-color: brown;
  width: 100px;
  height: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>

```



Flexbox Justify-Content

The CSS3 justify-content property is used to define the alignment along the main axis. It is a sub-property of CSS3 Flexbox layout module.

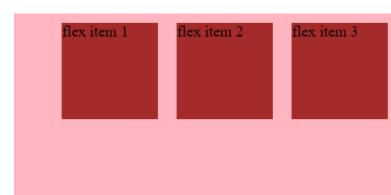
Its possible values are:

- **flex-start:** It is the default value. It sets the items at the beginning of the container
- **flex-end:** It sets the items at the end of the container.
- **Center:** It sets the items at the center of the container.
- **space-between:** It sets the items with space between the lines.
- **space-around:** It sets the items with space before, between, and after the lines.

```

<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  justify-content: flex-end;
  width: 400px;
  height: 200px;
  background-color: lightpink;
}
.flex-item {
  background-color: brown;
  width: 100px;
  height: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>

```



Flexbox align-items

The CSS3 flexbox align-items property is used to set the flexible container's items vertically align when the items do not use all available space on the cross-axis.

Its possible values are:

- **stretch:** It is the default value. It specifies that Items are stretched to fit the container.
- **flex-start:** It sets the items at the top of the container.
- **flex-end:** It sets the items at the bottom of the container.

- **center**: It sets the items at the center of the container (vertically).
- **baseline**: It sets the items at the baseline of the container.

Let's take some example to demonstrate the usage of above values.

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: -webkit-flex;
  display: flex;
  -webkit-align-items: stretch;
  align-items: stretch;
  width: 400px;
  height: 200px;
  background-color: lightpink;
}
.flex-item {
  background-color: brown;
  width: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>
```



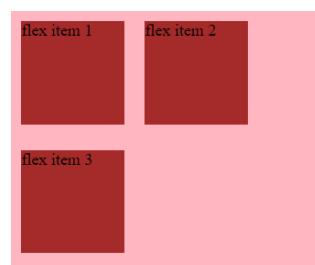
Flexbox flex-wrap

The CSS3 Flexbox **flex-wrap** property specifies if the flex-items should wrap or not, in the case of not enough space for them on one flex line.

Its possible values are:

- **nowrap**: It is the default value. The flexible items will not wrap.
- **wrap**: It specifies that the flexible items will wrap if necessary.
- **wrap-reverse**: It specifies that the flexible items will wrap, if necessary, in reverse order.

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-wrap: wrap;
  width: 300px;
  height: 250px;
  background-color: lightpink;
}
.flex-item {
  background-color: brown;
  width: 100px;
  height: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>
```



Flexbox align-content property

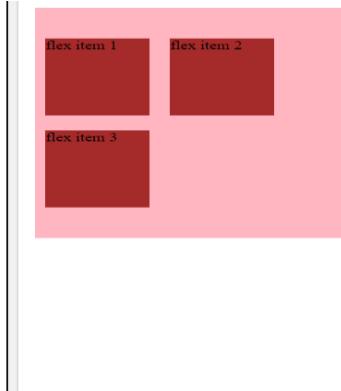
The CSS3 Flexbox align-content property is used to modify the behavior of the flex-wrap property. It is just like align-items, but it aligns flex lines instead of flex items.

Its possible values are:

- **stretch**: It is the default value. It specifies lines stretch to take up the remaining space.

- **flex-start:** It specifies that lines are packed toward the start of the flex container.
- **flex-end:** It specifies that lines are packed toward the end of the flex container.
- **center:** It specifies that lines are packed toward the center of the flex container.
- **space-between:** It specifies that lines are evenly distributed in the flex container.
- **space-around:** It specifies that lines are evenly distributed in the flex container, with half-size spaces on either end.

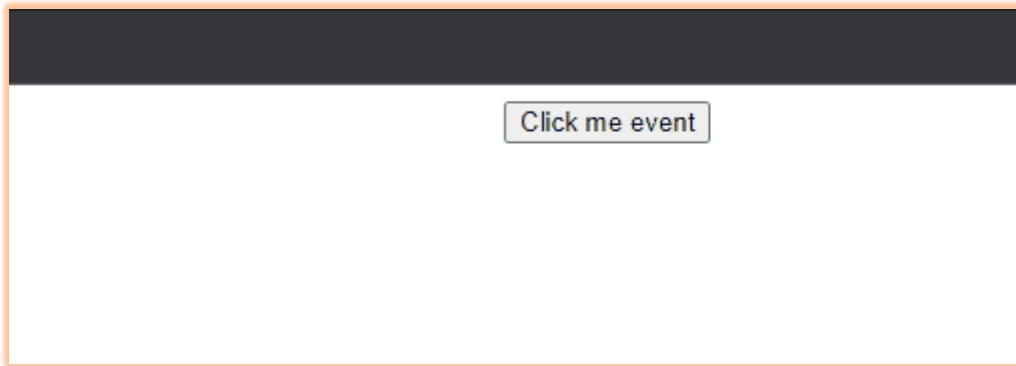
```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-wrap: wrap;
  align-content: center;
  width: 300px;
  height: 300px;
  background-color: lightpink;
}
.flex-item {
  background-color: brown;
  width: 100px;
  height: 100px;
  margin: 10px;
}
</style>
</head>
<body>
<div class="flex-container">
  <div class="flex-item">flex item 1</div>
  <div class="flex-item">flex item 2</div>
  <div class="flex-item">flex item 3</div>
</div>
</body>
</html>
```



1.2.3 Frontend Development with JavaScript

What Is JavaScript?

JavaScript is a scripting language for creating dynamic web page content. It creates elements for improving site visitors' interaction with web pages, such as dropdown menus, animated graphics, and dynamic background colors.



What Is JavaScript Used For?

JavaScript is used to make dynamic websites, web and mobile apps, games, web servers, back-end infrastructures, and more.

How to Use JavaScript in HTML

1. Inline JavaScript

Here, you have the JavaScript code in HTML tags in some special JS-based attributes.

For example, HTML tags have event attributes that allow you to execute some code inline when an event is triggered.

```
<button onclick="alert('You just clicked a button')">Click me!</button>
```

This is an example of inline JavaScript. The value of onclick can be some Match calculation, a dynamic addition to the DOM – any syntax-valid JavaScript code.

2. Internal JavaScript, with the script tag

Just like the style tag for style declarations within an HTML page, the script tag exists for JavaScript.

```
<script>
    function(){
        alert("I am inside a script tag")
    }
</script>
```

3. External JavaScript

You may want to have your JavaScript code in a different file.

```
<!-- index.html -->
<script src="./script.js"></script>
// script.js
alert("I am inside an external file");
```

The src attribute of the script tag allows you to apply a source for the JavaScript code. That reference is important because it notifies the browser to also fetch the content of script.js.

script.js can be in the same directory with index.html, or it can be gotten from another website. For the latter, you'll need to pass the full URL (<https://.../script.js>).

Variable Keywords

KEYWORD	SCOPE	REDECLARATION & REASSIGNMENT	HOISTING
var	Global, Local	yes & yes	yes, with default value
let	Global, Local, Block	no & yes	yes, without default value
const	Global, Local, Block	no & no	yes, without default value

Data Types

```
const x = 5;
const y = "Hello";
```

Here, 5 is an integer data and "Hello" is a string data.

There are eight basic data types in JavaScript. They are:

Data Types	Description	Example
<code>String</code>	represents textual data	<code>'hello'</code> , <code>"hello world!"</code> etc
<code>Number</code>	an integer or a floating-point number	<code>3</code> , <code>3.234</code> , <code>3e-2</code> etc.
<code>BigInt</code>	an integer with arbitrary precision	<code>900719925124740999n</code> , <code>1n</code> etc.
<code>Boolean</code>	Any of two values: true or false	<code>true</code> and <code>false</code>
<code>undefined</code>	a data type whose variable is not initialized	<code>let a;</code>
<code>null</code>	denotes a <code>null</code> value	<code>let a = null;</code>
<code>Symbol</code>	data type whose instances are unique and immutable	<code>let value = Symbol('hello');</code>
<code>Object</code>	key-value pairs of collection of data	<code>let student = {};</code>

- **JavaScript String**

String is used to store text. In JavaScript, strings are surrounded by quotes:

- Single quotes: 'Hello'
- Double quotes: "Hello"
- Backticks: `Hello`

Single quotes and double quotes are practically the same and you can use either of them.

Backticks are generally used when you need to include variables or expressions into a string. This is done by wrapping variables or expressions with \${variable or expression}

- **JavaScript Number**

Number represents integer and floating numbers (decimals and exponentials). A number type can also be `+Infinity`, `-Infinity`, and `NaN` (not a number).

- **JavaScript BigInt**

In JavaScript, Number type can only represent numbers less than $(2^{53} - 1)$ and more than $-(2^{53} - 1)$. However, if you need to use a larger number than that, you can use the BigInt data type.

A BigInt number is created by appending `n` to the end of an integer.

- **JavaScript Boolean**

This data type represents logical entities. Boolean represents one of two values: true or false.

- **JavaScript undefined**

The undefined data type represents value that is not assigned. If a variable is declared but the value is not assigned, then the value of that variable will be undefined

- **JavaScript null**

In JavaScript, null is a special value that represents empty or unknown value.

- **JavaScript Symbol**

This data type was introduced in a newer version of JavaScript (from ES2015). A value having the data type Symbol can be referred to as a symbol value. Symbol is an immutable primitive value that is unique

- **JavaScript Object**

An object is a complex data type that allows us to store collections of data.

JavaScript typeof

To find the type of a variable, you can use the `typeof` operator.

Example Code:

```
<html>
  <body>
    <script>
      //strings-example
      const namea = 'EDUNETFOUNDATION';
      const name1 = "TECHSAKSHAM PROGRAM";
      console.log(`The names are ${namea} and ${name1}`);
      //number-example
      const number4 = -3;
      const number5 = -3.433;
      const number6 = -3e5 // -3 * 10^5
      const number1 = -3/0;
      console.log(number1); // -Infinity
      const number2 = -3/0;
      console.log(number2); // -Infinity
      console.log(number4 + "  
" + number5 + "  
" + number6 + "  
" + number1 + "  
" + number2);
      //NaN-example
      // strings can't be divided by numbers
      const number3 = "abc" / 3;
      console.log(number3); // NaN
      //BigInt-value-example
      const value3 = 900719925124740998n;
      console.log(value3);
      //Adding two big integers
      const result1 = value3 + 1n;
      console.log(result1); // 900719925124740999n
      const value4 = 900719925124740998n;
      //Error! BigInt and number cannot be added
      console.log(value4);
      const result2 = value4 + 1n;
      console.log(result2);
      //Boolean-example
      const dataChecked = true;
      const valueCounted = false;
      let nameb;
      console.log(nameb); // undefined
      let namec = undefined;
      console.log(namec); // undefined
      const numberg = null;
      console.log(nameb + "  
" + namec);
      //Symbol-example
      const value1 = Symbol('hello');
      const value2 = Symbol('hello');
      //Object-example
      const student = {
        firstName: 'ram',
        lastName: null,
        class: 10
      };
      console.log(student.firstName + student.lastName + student.class);
      //typeof-examples
      const name = 'ram';
      console.log(typeof(name)); // returns "string"
      const number = 4;
      console.log(typeof(number)); // returns "number"
      const valueChecked = true;
      console.log(typeof(valueChecked)); // returns "boolean"
      const a = null;
      console.log(typeof(a)); // returns "object"
    </script>
  </body>
</html>
```

Output:

The names are EDUNETFOUNDATION and TECHSAKSHAM PROGRAM	index.html:8
Infinity	index.html:14
-Infinity	index.html:16
3 3.433 300000 Infinity -Infinity	index.html:17
NaN	index.html:21
900719925124740998n	index.html:24
900719925124740999n	index.html:27
900719925124740998n	index.html:30
900719925124740999n	index.html:32
undefined	index.html:37
undefined	index.html:39
undefined undefined	index.html:41
ram null 10	index.html:51
string	index.html:54
number	index.html:56
boolean	index.html:58
object	index.html:60

JS Console

JavaScript console.log(): All modern browsers have a web console for debugging. The console.log() method is used to write messages to these consoles.

For example,

```
let sum = 44;
console.log(sum); // 44
```

When you run the above code, 44 is printed on the console.

A screenshot of a browser's developer tools console. On the left, the DOM tree shows a script element containing the code: `let sum = 44;` and `console.log(sum); // 44`. On the right, the console output shows the number 44 and the message "Live reload enabled.".

```
x.html > html > body > script
<!Doctype html>
<html>
|   <body>
|   |   <script>
let sum = 44;
console.log(sum); // 44

</script>
</body>
</html>
```

JS Dialogue Boxes

Dialogue boxes are a kind of popup notification, this kind of informative functionality is used to show success, failure, or any particular/important notification to the user. JavaScript uses 3 kinds of dialog boxes: Alert, Prompt, and Confirm

- **Alert Box:** An alert box is used on the website to show a warning message to the user that they have entered the wrong value other than what is required to fill in that position. Nonetheless, an alert box can still be used for friendlier messages. The alert box gives only one button "OK" to select and proceed.

Example Code:

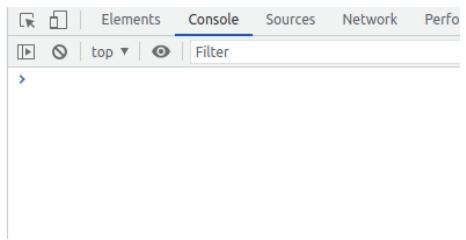
```

<!Doctype html>
<html>
  <body>
    <script>
      function Warning() {
        alert ("Warning danger you have not filled everything");
        console.log ("Warning danger you have not filled everything");
      }
    </script>
    <p>Click the button to check the Alert Box functionality</p>
    <form>
      <input type="button" value="Click Me" onclick="Warning();"/>
    </form>
  </script>
</body>
</html>

```

Output:

Click the button to check the Alert Box functionality



- **Confirm box:** A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks on the OK button, the window method `confirm()` will return true. If the user clicks on the Cancel button, then `confirm()` returns false and will show null.

Example Code:

```

<!Doctype html>
<html>
  <body>
    <script>
      function Confirmation(){
        var Val = confirm("Do you want to continue ?");
        if (Val == true) {
          console.log(" CONTINUED!");
          return true;
        } else {
          console.log("NOT CONTINUED!");
          return false;
        }
      }
    </script>
    <p>Click the button to check the Alert Box functionality</p>
    <form>
      <input type="button" value="Click Me" onclick="Confirmation();"/>
    </form>
  </script>
</body>
</html>

```

Output:



- **Prompt Box:** A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks the OK button, the window method `prompt()` will return the entered value from the text box. If the user clicks the Cancel button, the window method `prompt()` returns null.

Example Code:

```
<!Doctype html>
<html>
  <body>
    <button type="button" onclick="Value()">Click</button>
    <script>
      function Value(){
        var Val = prompt("Enter your name : ", "Please enter your name");
        console.log("You entered : " + Val);
      }
    </script>
  </body>
</html>
```

Output:

The screenshot shows a browser window with a 'Click' button. When the button is clicked, a prompt box appears asking for a name. The user types 'EDUNET FOUNDATION' into the prompt box. The browser's developer tools console is open at the bottom, showing the following output:

```
127.0.0.1:5500 says
Enter your name :
EDUNET FOUNDATION
OK Cancel
Live reload enabled.
You entered : EDUNET FOUNDATION
```

JS Operator

In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables).

For Example,

`2 + 3; // 5`

Here `+` is an operator that performs addition, and `2` and `3` are operands.

Operator Type

- Assignment Operators
- Arithmetic Operators

- Comparison Operators
- Logical Operators
- Bitwise Operators
- String Operators
- Other Operators
- Assignment Operators

Assignment operators are used to assign values to variables.

For example, const x = 5;

Here, the = operator is used to assign value 5 to variable x.

Here's a list of commonly used assignment operators:

Arithmetic operators are used to perform arithmetic calculations.

For example, const number = 3 + 5; // 8

Here, the + operator is used to add two operands.

Comparison operators compare two values and return a boolean value, either true or false.

For example, const a = 3, b = 2;

```
    console.log(a > b); // true
```

Here, the comparison operator > is used to compare whether a is greater than b.

Logical operators perform logical operations and return a boolean value, either true or false.

For example, const x = 5, y = 3;

```
    (x < 6) && (y < 5); // true
```

Here, && is the logical operator AND. Since both x < 6 and y < 5 are true, the result is true.

Bitwise operators perform operations on binary representations of numbers.

Bitwise AND (&): The bitwise AND (&) operator returns a number or BigInt whose binary representation has a 1 in each bit position for which the corresponding bits of both operands are 1.

```
const a = 5;      // 0000000000000000000000000000000101
```

```
const b = 3;      // 0000000000000000000000000000000011
```

```
console.log(a & b); // 0000000000000000000000000000000001
```

// Expected output: 1

String Operators

In JavaScript, you can also use the + operator to concatenate (join) two or more strings.

Example Code:

```

<!DOCTYPE html>
<html>
  <body>
    <script>
      let x = 5;
      let y = 3;
      // addition
      console.log('x + y = ', x + y); // 8
      // subtraction
      console.log('x - y = ', x - y); // 2
      // multiplication
      console.log('x * y = ', x * y); // 15
      // division
      console.log('x / y = ', x / y); // 1.6666666666666667
      // remainder
      console.log('x % y = ', x % y); // 2
      // increment
      console.log('++x = ', ++x); // x is now 6
      console.log('x++ = ', x++); // prints 6 and then increased to 7
      console.log('x = ', x); // 7
      // decrement
      console.log('--x = ', --x); // x is now 6
      console.log('x-- = ', x--); // prints 6 and then decreased to 5
      console.log('x = ', x); // 5
      // exponentiation
      console.log('x ** y = ', x ** y);
      // equal operator
      console.log(2 == 2); // true
      console.log(2 == '2'); // true
      // not equal operator
      console.log(3 != 2); // true
      console.log('hello' != 'Hello'); // true
      // strict equal operator
      console.log(2 === 2); // true
      console.log(2 === '2'); // false
      // strict not equal operator
      console.log(2 !== '2'); // true
      console.log(2 !== 2); // false
      // logical AND
      console.log(true && true); // true
      console.log(true && false); // false
      // logical OR
      console.log(true || false); // true
      // logical NOT
      console.log(!true); // false
      // concatenation operator
      console.log('hello' + 'world');
      let a = 'JavaScript';
      a += ' tutorial'; // a = a + ' tutorial';
      console.log(a);
    </script>
  </body>
</html>

```

Output:

```

x + y = 8
x - y = 2
x * y = 15
x / y = 1.6666666666666667
x % y = 2
++x = 6
x++ = 6
x = 7
--x = 6
x-- = 6
x = 5
x ** y = 125
true
true
true
true
false
true
true
false
true
false
false
helloworld
JavaScript tutorial

```

Default levels ▾ | 1 Issue | P 1 | ⓘ

index.html:8
index.html:10
index.html:12
index.html:14
index.html:16
index.html:18
index.html:19
index.html:20
index.html:22
index.html:23
index.html:24
index.html:26
index.html:28
index.html:29
index.html:31
index.html:32
index.html:34
index.html:35
index.html:37
index.html:38
index.html:40
index.html:41
index.html:43
index.html:45
index.html:47
index.html:50

JS Functions

A function is a set of statements that take inputs, do some specific computation, and produce output. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call that function.

Basic javascript function,

```
function myFunction(g1, g2) {
```

```
  return g1 / g2;
}
```

```
const value = myFunction(8, 2); // Calling the function
```

```
console.log(value);
```

Output: 4

Syntax: The basic syntax to create a function in JavaScript is shown below.

```
function functionName(Parameter1, Parameter2, ...)  
{  
    // Function body  
}
```

To create a function in JavaScript, we have to first use the keyword function, separated by the name of the function and parameters within parenthesis. The part of the function inside the curly braces {} is the body of the function.

Function Definition: Before, using a user-defined function in JavaScript we have to create one. We can use the above syntax to create a function in JavaScript. A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

- Every function should begin with the keyword function followed by,
- A user-defined function name that should be unique,
- A list of parameters enclosed within parentheses and separated by commas,
- A list of statements composing the body of the function enclosed within curly braces {}.

Function Declaration: It declares a function with a function keyword. The function declaration must have a function name.

Syntax:

```
function geeksforGeeks(paramA, paramB) {  
    // Set of statements  
}
```

Function Expression: It is similar to a function declaration without the function name. Function expressions can be stored in a variable assignment.

Syntax:

```
let a= function(paramA, paramB) {  
    // Set of statements  
}
```

Example Code & Output:

```

<!DOCTYPE html>
<html>
<body>
    <h1>Function Expression</h1>
    <p id="p1"></p>

    <script>
        function Sum(num1, num2) {
            return num1 + num2;
        }

        var result = Sum(10,20);

        document.getElementById("p1").innerHTML = result;
    </script>
</body>
</html>

```

Function Expression

30

Arrow Function: It is one of the most used and efficient methods to create a function in JavaScript because of its comparatively easy implementation. It is a simplified as well as a more compact version of a regular or normal function expression or syntax.

Syntax:

```
let function_name = (argument1, argument2 ,..) => expression
```

Example Code & Output:

```

<!DOCTYPE html>
<html>
<body>
    <h1>ARROW FUNCTION</h1>
    <p id="p1"></p>

    <script>
        let square = num => num * num;

        let result = square(5);

        document.getElementById("p1").innerHTML = result;
    </script>
</body>
</html>

```

ARROW FUNCTION

25

Function Parameters: Parameters are additional information passed to a function. A function in JavaScript can have any number of parameters and also at the same time, a function in JavaScript can not have a single parameter.

Example Code & Output:

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Function Parameters</h1>

    <script>
        function greet(firstName, lastName) {
            alert("Hello " + firstName + " " + lastName);
        }

        greet("Bill", "Gates");
        greet(100, 200);
    </script>
</body>
</html>

```



```

<!DOCTYPE html>
<html>
<body>
    <h1> Function Parameters</h1>

    <script>
        function greet(firstName, lastName) {
            alert("Hello " + firstName + " " + lastName);
        }

        greet("Bill", "Gates");
        greet(100, 200);
    </script>
</body>
</html>

```

Function Parameters

Calling Functions: After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between the parenthesis and a semicolon at the end. The below syntax shows how to call functions in JavaScript:

Syntax:

```
functionName( Value1, Value2, ...);
```

Example Code & Output:

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript function</h1>

    <script>
        function greet() {
            alert("Hello World!");
        }

        greet();
    </script>
</body>
</html>

```

Return Statement: There are some situations when we want to return some values from a function after performing some operations.

Syntax: The most basic syntax for using the return statement is:
return value;

Example Code & Output:

Return Value from a function

```

<!DOCTYPE html>
<html>
<body>
    <h1>Return Value from a function</h1>
    <p id="p1"></p>

    <script>
        function getNumber() {
            return 10;
        }

        let result = getNumber();

        document.getElementById("p1").innerHTML = result;
    </script>
</body>
</html>

```

10

The return statement begins with the keyword *return* separated by the value which we want to return from it. We can use an expression also instead of directly returning the value.

Anonymous Function: create anonymous functions, which are functions without a name. Anonymous functions are often used as arguments to other function. Anonymous functions are typically used in functional programming e.g. callback function, creating closure or immediately invoked function expression.

Example Code & Output:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Anonymous Function as Callback</h1>
    <p id="p1"></p>

    <script>
        let numbers = [2, 3, 4, 5];

        let squareNumbers = numbers.map(function(number) {
            return number * number;
        });

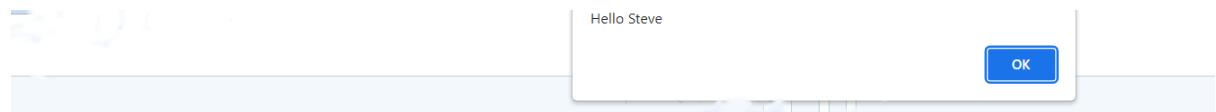
        document.getElementById("p1").innerHTML = squareNumbers;
    </script>
</body>
</html>
```

Anonymous Function as Callback

4,9,16,25

Nested Functions: A function can have one or more inner functions. These nested functions are in the scope of outer function. Inner function can access variables and parameters of outer function. However, outer function cannot access variables defined inside inner functions.

Example Code & Output:



```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Nested Function</h1>

    <script>
        function ShowMessage(firstName)
        {
            function SayHello() {
                alert("Hello " + firstName);
            }

            return SayHello();
        }

        ShowMessage("Steve");
    </script>
</body>
</html>
```

Demo: Nested Function

Control Statement

Control statements are intended to allow users to write scripts that determine which lines of code get evaluated and how many times they get evaluated. Control statements are classified into two types.

Types of Control Statements in JS

There are basically two types of control statements in Javascript. Namely, Conditional Statements and Iterative Statements (in simple words, Loops).

- **Conditional Statements:** These are the ones that make a decision. There is an expression that gets passed, and then the conditional statement decides on it. It is either YES or NO.
- **Iterative Statements (Loop):** This is a type of statement that keeps repeating itself till a condition is satisfied. In simple words, if we have an expression, until and unless it gets satisfied, the statement will keep repeating itself.

Different types of Conditional Statements

If Statement

As the name suggests, we use the If Statement if we want to check for a specific condition. When we are using the IF condition, the inner code block gets executed only when the given condition is satisfied.

Syntax:

```
if (condition_is_given_here) {  
    // If the condition is met, the code block will get executed.  
}
```

If-Else Statement

The If-Else Statement is an extended version of the If statement. We use this one when we want to check a condition we want and another one if the evaluation turns out otherwise.

Syntax:

```
if (condition_is_given_here) {  
    // If the condition is met, this code block will get executed.  
}  
else {  
    /*If the condition is not met and turns out to be false, this code block will get  
    executed. */  
}
```

As we can see, the first block of code gets executed when the condition in IF-ELSE is satisfied. The second block of code gets executed when the condition does not get satisfied.

Switch Statement

A switch statement is similar to an IF statement. It is used when the user needs to execute one code block execution from their own choices out of several code block execution options. It is based on the outcome of the expression given. Switch statements hold an expression that is compared to the values of the following cases. If a match is found, the code associated with that case is executed.

Syntax:

```

switch (expression) {
    case first:
        // Here, we add the code block that is to be executed
        Break;
    // Break creates a separation between each case
    case second:
        // Here, we add the code block that is to be executed
        Break;
    case third:
        // Here, we add the code block that is to be executed
        Break;
    default:
        // Here, we add the default code block that is to be executed. It happens if none of
        // the above cases is executed.
}

```

Different types of Iterative Statements

While Statement

One of the control flow statements is the While Statement. When the condition is met, they run a code block. However, unlike the IF statement, while keeps repeating itself until the condition is met. The difference between IF and while is that IF executes code when the condition is met. Whereas while statement keeps repeating itself till the condition is met.

Syntax:

```

while (the_condition_is_put_here)
{
    // The code block is executed when the conditions are met
}

```

Do-While Statement

We can call the Do-While Statement a sibling of the While Statement. Similar to a while loop, but with the addition of a condition at the end of the loop. DO-WHILE, also known as an Exit Control Loop, performs the code and then checks for the condition.

Syntax:

```

while
{
    // Code block gets executed till the condition is satisfied
} (we_add_the_conditon_here)

```

The loop will get repeated if the condition at the end is met.

For Statement

Now let us take a look at the For Loop. A for loop will run a code block several times. The FOR statement is shorter and easier to debug than other loops. This is because it contains initialization, condition, and increment or decrement on a single line.

Syntax:

```
for (here_we_put initialize; condition; increment/decrement)
{
// Here_we_put_the_code_block_that_we_need_to_execute
}
```

Example Code & Output:

```
<html>
<body>
<script>
... var numbers = [65, 99, 34, 12, 10, 77]; //array of numbers
... for(number of numbers){ //iterating through the values of numbers
...   if(number % 5 == 0){ //chhecking if the number is divisible by 5
...     document.write(number + " is divisible by 5.<br>")
...   }else{
...     continue; //restarts the loop
...     document.write(number + " is not divisible by 5.<br>") //skipped statement
...   }
...
</script>
</body>
</html>
```

< 65 is divisible by 5.
10 is divisible by 5.

Document Object Model(DOM)

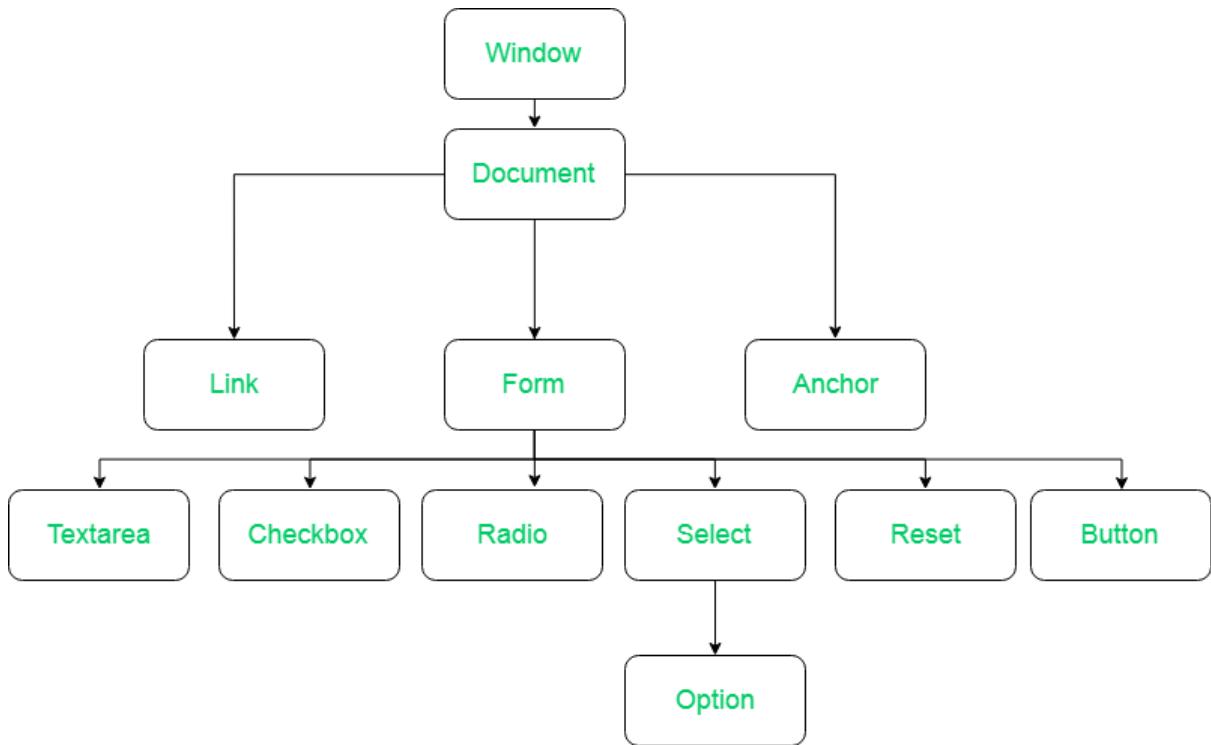
The Document Object Model (DOM) is a programming interface for HTML (HyperText Markup Language) and XML(Extensible markup language) documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

Why DOM is required?

HTML is used to structure the web pages and Javascript is used to add behavior to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). DOM is basically the representation of the same HTML document but in a different format with the use of objects. Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

Structure of DOM: DOM can be thought of as a Tree or Forest(more than one tree). The term structure model is sometimes used to describe the tree-like representation of a document. Each branch of the tree ends in a node, and each node contains objects. Event listeners can be added to nodes and triggered on an occurrence of a given event.

Properties of DOM



- **Window Object:** Window Object is object of the browser which is always at top of the hierarchy. It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.
- **Document object:** When an HTML document is loaded into a window, it becomes a document object. The 'document' object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the 'document' object. Document object is property of window object.
- **Form Object:** It is represented by *form* tags.
- **Link Object:** It is represented by *link* tags.
- **Anchor Object:** It is represented by *a href* tags.
- **Form Control Elements:** Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

Methods of Document Object:

`write("string")`: Writes the given string on the document.

```
<!DOCTYPE html>
<html>
<body>
<h1>Write Method</h1>
<script>
document.write("EDUNET FOUNDATION")
</script>
</body>
</html>
```

Write Method

EDUNET FOUNDATION

`getElementById()`: returns the element having the given id value.

```

<!DOCTYPE html>
<html>
<body>

<h1 id="a">The Document Object</h1>
<h2>The getElementById() Method</h2>

<script>
document.getElementById("a").style.color = "red";
</script>

</body>
</html>

```

The Document Object

The getElementById() Method

getElementsByName(): returns all the elements having the given name value.

```

<!DOCTYPE html>
<html>
<body>
<h2>The document.getElementsByName Method</h2>
Cats:
<input name="animal" type="checkbox" value="Cats">
Dogs:
<input name="animal" type="checkbox" value="Dogs">
<p>Number of elements with name="animal" is:</p>
<p id="demo"></p>
<script>
let num = document.getElementsByName("animal").length;
document.getElementById("demo").innerHTML = num;
</script>
</body>
</html>

```

The document.getElementsByName Method

Cats: Dogs:

Number of elements with name="animal" is:

2

getElementsByTagName(): returns all the elements having the given tag name.

```

<!DOCTYPE html>
<html>
<body>
<h2>The getElementsByTagName() Method</h2>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<script>
document.getElementsByTagName("p")[0].innerHTML = "Hello World!";
</script>
</body>
</html>

```

The getElementsByTagName() Method

Hello World!

This is a paragraph.

This is a paragraph.

getElementsByClassName(): returns all the elements having the given class name.

```

<!DOCTYPE html>
<html>
<head>
<style>
div {
    border: 1px solid black;
    padding: 8px;
}
</style>
</head>
<body>
<h2>The getElementsByClassName() Method</h2>
<p>Change the background color of the first element with the classes "example" and "color":</p>
<div class="example">
<p>This is a paragraph.</p>
</div>
<br>
<div class="example color">
<p>This is a paragraph.</p>
</div>
<br>
<div class="example color">
<p>This is a paragraph.</p>
</div>
<script>
const collection = document.getElementsByClassName("example color");
collection[0].style.backgroundColor = "red";
</script>
</body>
</html>

```

The getElementsByClassName() Method

Change the background color of the first element with the classes "example" and "color":

This is a paragraph.

This is a paragraph.

This is a paragraph.

Realtime Example:

```

<!DOCTYPE html>
<html>
<head>
<title>DOM manipulation</title>
</head>
<body>
<label>Enter Value 1: </label>
<input type="text" id="val1" />
<br />
<label>Enter Value 2: </label>
<input type="text" id="val2" />
<br />
<button onclick="getAdd()">Click To Add</button>
<p id="result"></p>
<script type="text/javascript">
function getAdd() {
    // Fetch the value of input with id val1
    const num1 = Number(document.getElementById("val1").value);
    // Fetch the value of input with id val2
    const num2 = Number(document.getElementById("val2").value);
    const add = num1 + num2;
    console.log(add);
    // Displays the result in paragraph using dom
    document.getElementById("result").innerHTML = "Addition : " + add;
    // Changes the color of paragraph tag with red
    document.getElementById("result").style.color = "red";
}
</script>
</body>
</html>

```

Enter Value 1:

Enter Value 2:

Enter Value 1:

Enter Value 2:

Enter Value 1:

Enter Value 2:

Addition : 100

JavaScript Objects

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.// object

```

const student = {
    firstName: 'ram',
    class: 10
};

```

Here, student is an object that stores values such as strings and numbers.

JavaScript Object Declaration

The syntax to declare an object is:

```

const object_name = {
    key1: value1,
    key2: value2
}

```

Here, an object object_name is defined. Each member of an object is a key: value pair separated by commas and enclosed in curly braces {}.



```
<!DOCTYPE html>
<html>
<head>
    <title>DOM manipulation</title>
</head>
<body>
    <p id="result"></p>
    <script type="text/javascript">
        // object creation
        const person = {
            name: 'John',
            age: 20
        };
        console.log(person); // object
    </script>
</body>
</html>
```

JavaScript Object Properties

In JavaScript, "key: value" pairs are called properties. For example,

```
let person = {
    name: 'John',
    age: 20
};
```

Here, name: 'John' and age: 20 are properties.

Accessing Object Properties

You can access the value of a property by using its key.

1. Using dot Notation

Here's the syntax of the dot notation.

objectName.key

2. Using bracket Notation

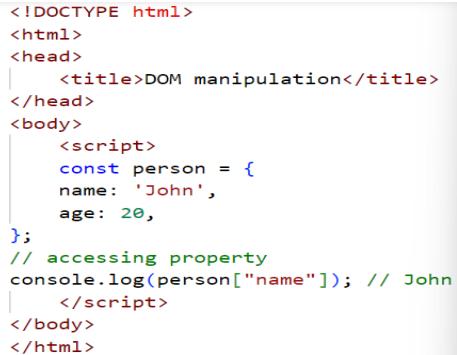
Here is the syntax of the bracket notation.

objectName["propertyName"]

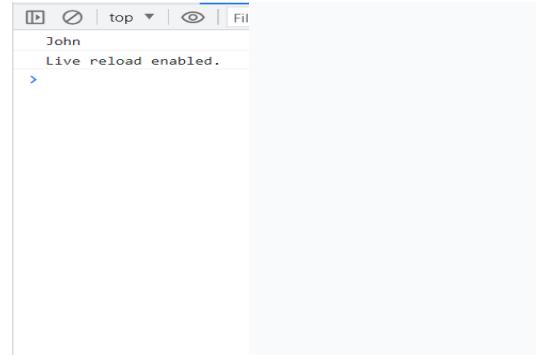
JavaScript Nested Objects

An object can also contain another object.

For example,



```
<!DOCTYPE html>
<html>
<head>
    <title>DOM manipulation</title>
</head>
<body>
    <script>
        const person = {
            name: 'John',
            age: 20,
        };
        // accessing property
        console.log(person["name"]); // John
    </script>
</body>
</html>
```



The screenshot shows two panels from a browser's developer tools. The left panel displays two snippets of JavaScript code. The top snippet creates a student object with nested marks and prints its properties. The bottom snippet creates a person object with a greet function and prints its value. The right panel shows the resulting object structure in the 'Elements' tab and the output of the console.log statements in the 'Console' tab, which includes the object's properties and their values.

```

<html>
<head>
|   <title>DOM manipulation</title>
</head>
<body>
|   <script>
// nested object
const student = {
  name: 'John',
  age: 20,
  marks: {
    science: 70,
    math: 75
  }
}
// accessing property of student object
console.log(student.marks); // {science: 70, math: 75}
// accessing property of marks object
console.log(student.marks.science); // 70
|   </script>
</body>
</html>

<html>
<head>
|   <title>DOM manipulation</title>
</head>
<body>
|   <script>
const person = {
  name: 'Sam',
  age: 30,
  // using function as a value
  greet: function() { console.log('hello') }
}

person.greet(); // hello
|   </script>
</body>
</html>

```

Object Methods

In JavaScript, an object can also contain a function.

Access nested JSON objects

Accessing nested **json** objects is just like accessing nested arrays. **Nested objects** are the objects that are inside an another object.

The screenshot shows a browser window with a script that defines a person object with a vehicles property containing an array of vehicle names. The script then uses document.write to output a message including the vehicles property. The right side of the image shows the resulting output: "Mr Ram has a car called limousine".

```

index.html / ⑤ num1
<html>
<body>
<script>
var person = {
  "name": "Ram",
  "age": 27,
  "vehicles": [
    "car": "limousine",
    "bike": "ktm-duke",
    "plane": "lufthansa"
  ]
}
document.write("Mr Ram has a car called" + " " + person.vehicles[0]);
</script>
</body>
</html>

```

Mr Ram has a car called limousine

Introduction to the Node Object:

Node represents a point within a DOM; this could be an element, text, a comment, and more.

HTMLHeadingElement

The `HTMLHeadingElement` interface represents the different heading elements, `<h1>` through `<h6>`. It inherits methods and properties from the `HTMLElement` interface.



In the image we can see that the `HTMLHeadingElement` object (and all other HTML element objects like `HTMLLIElement`, `HTMLAnchorElement`, and so on) inherits from 4 other objects:

- `HTMLElement`
- `Element`
- `Node`
- `EventTarget`

HTMLElement represents any HTML element in the HTML DOM. This is a generic object type for more specific objects like `HTMLLIElement`, `HTMLImageElement`, and so on.

Element represents any element in a DOM. This is even more generic than `HTMLElement`, because this can include HTML elements in the HTML DOM, or SVG elements in an SVG DOM.

EventTarget represents any object that can be the target of events and that we can attach an event listener to.

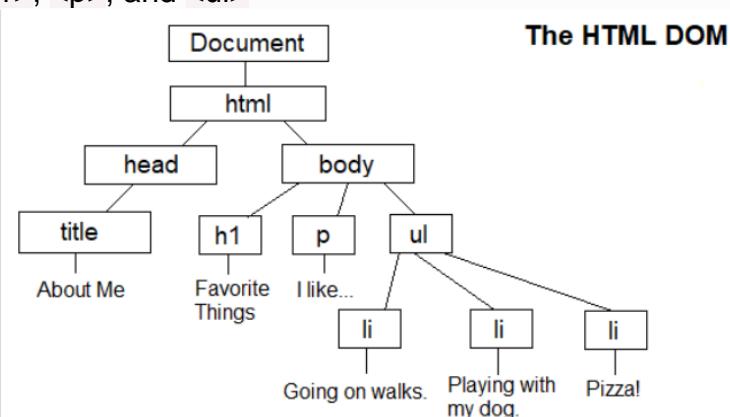
Node as an even more generic representation of the `Element` object, but to get to the heart of it, we need to understand what a node in a tree is.

Visualization of an HTML DOM Tree

In fact the terms "parent", "child or "children", and "siblings" are how we describe the hierarchical relationships in the DOM tree. In the image below, we can list out a few relationships:

- `<body>` is the child of `<html>`, and `<html>` is the parent of `<body>`
- The three `` elements are all siblings, as well as children of the `` element.
- `<body>` is the parent of `<h1>`, `<p>`, and ``

```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en-US">
3    <head>
4      <title>About Me</title>
5    </head>
6    <body>
7      <h1>Favorite Things</h1>
8      <p>I like...</p>
9      <ul>
10        <li>Going on walks.</li>
11        <li>Playing with my dog.</li>
12        <li>Pizza!</li>
13      </ul>
14    </body>
15  </html>
```



Visualization of Node Types

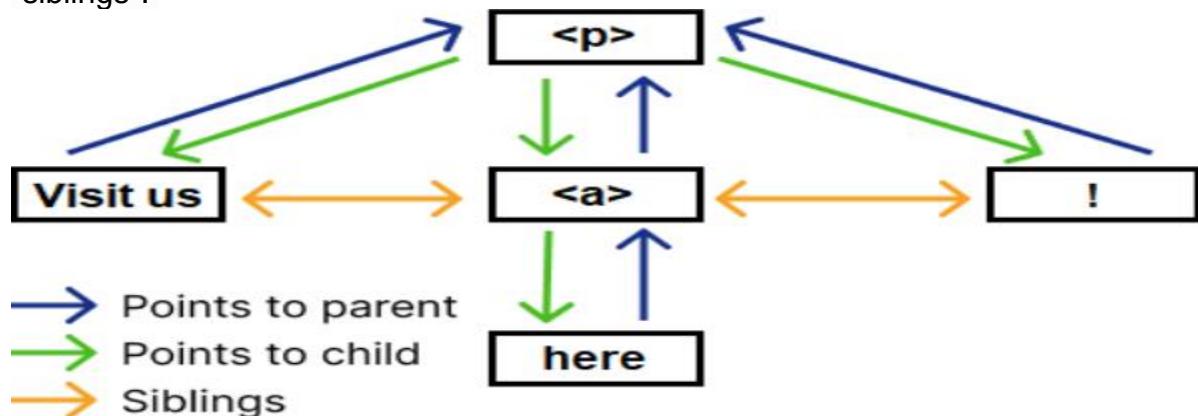
A **node type** is just like an object type, having a name and representing a specific type of node in the DOM tree.

```
<p id="test">Visit us <a href="www.example.com">here</a>!</p>
```

```
<p id="test">Visit us <a href="www.example.com">here</a>!</p>
<p id="test"></p>           <!-- An element node
   |
   +-- Visit us            <!-- A text node
   |
   +-- <a href="www.example.com"></a>    <!-- An element node
       |
       +-- here             <!-- A text node
   |
   +-- !                  <!-- A text node
```

Visualization of Node Relationships

Relationships between nodes and how we describe them with "parent", "child", and "siblings".



Here are a few relationships between the nodes in the above image:

- The `<p>` element node is the parent of the text node "Visit us", the element node `<a>`, and the text node "!".
- The text node "Visit us", the element node `<a>`, and the text node "!" are all siblings.
- The text node "here" is the child of the element node `<a>`.

Node Properties and Methods

- The data structure of the DOM is a tree, which is a hierarchical collection of nodes.
- Each node in the DOM is an object. Generally speaking, a node in a tree can be any value, but for the DOM they are all objects.
- The `Node` object type represents a node within the DOM.
- There are many node types, each of which serves to distinguish the nodes in a tree. Common ones are element nodes, text nodes, and comment nodes.
- `Node` is the most generic object type that represents node in the DOM tree, and there are other objects that represent specific node types.

Creating Nodes

There's a couple ways to create nodes, each of which belongs to the document object:

- `document.createComment()`
- `document.createTextNode()`
- `document.createAttribute()`

Notably `document.createElement()` does not return a node object.

If we wanted to create an attribute or text, we'd use these methods instead:

- Setting `HTMLElement.innerText` to change the value of the text for an element.
- Accessing attribute properties or calling `Element.setAttribute()` method.

Adding New Nodes

A few Node methods to add new nodes to the DOM are:

- `Node.appendChild()`
- `Node.insertBefore()`
- `Node.removeChild()`

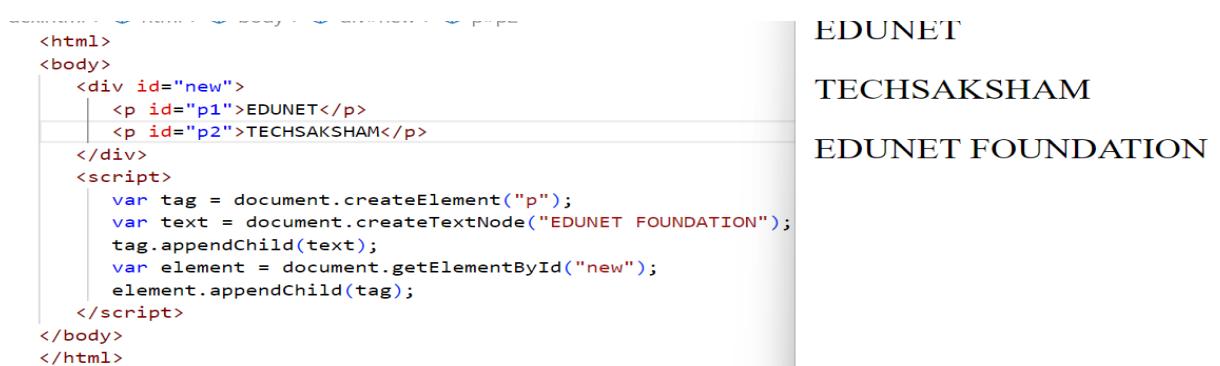
DOM Traversal

To move along the DOM, accessing parent, child, sibling elements, we might use the following Node properties.

- `Node.childNodes`
- `Node.firstChild`
- `Node.nextSibling`
- `Node.previousSibling`
- `Node.parentNode`
- `Node.textContent`
- `Node.nodeName`

Example1

In the following, initially the div section consists of only 2 texts. But later on, one more text is created and added to the div section as shown in the output.



The screenshot shows a browser window with developer tools open. On the left, the DOM tree is displayed with a `<div id="new">` node containing two `<p>` elements: `p1` with text "EDUNET" and `p2` with text "TECHSAKSHAM". A `<script>` block is also present. On the right, the page content is shown with the text "EDUNET", "TECHSAKSHAM", and "EDUNET FOUNDATION" stacked vertically.

```
<html>
  <body>
    <div id="new">
      <p id="p1">EDUNET</p>
      <p id="p2">TECHSAKSHAM</p>
    </div>
    <script>
      var tag = document.createElement("p");
      var text = document.createTextNode("EDUNET FOUNDATION");
      tag.appendChild(text);
      var element = document.getElementById("new");
      element.appendChild(tag);
    </script>
  </body>
</html>
```

Example 2

The following is an example program on how to add an element to HTML DOM.

```

<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript HTML DOM</h2>
    <p>How to add a new element to HTML DOM</p>
    <div id="div1">
        <p id="p1">Introduction.</p>
        <p id="p2">Conclusion.</p>
    </div>
    <script>
        const para = document.createElement("p");
        const node = document.createTextNode("The end.");
        para.appendChild(node);
        const element = document.getElementById("div1");
        const child = document.getElementById("p2");
        element.appendChild(para,child);
    </script>
</body>
</html>

```

Example 3

The following is an example program on how to add an element to HTML DOM. Here, the insertBefore method is used instead of append method to add the element to the div tag.

```

<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript HTML DOM</h2>
    <p>How to add a new element to HTML DOM</p>
    <div id="div1">
        <p id="p1">Introduction.</p>
        <p id="p2">Conclusion.</p>
    </div>
    <script>
        const para = document.createElement("p");
        const node = document.createTextNode("To begin with..."); // New element
        para.appendChild(node);
        const element = document.getElementById("div1");
        const child = document.getElementById("p2");
        element.insertBefore(para,child); // Insert before p2
    </script>
</body>

```

JS Events

An event is a signal that something has happened. All DOM nodes generate such signals (but events are not limited to DOM).

Here's a list of the most useful DOM events, just to take a look at:

Mouse events:

- click – when the mouse clicks on an element (touchscreen devices generate it on a tap).
- contextmenu – when the mouse right-clicks on an element.
- mouseover / mouseout – when the mouse cursor comes over / leaves an element.
- mousedown / mouseup – when the mouse button is pressed / released over an element.
- mousemove – when the mouse is moved.

Keyboard events:

- keydown and keyup – when a keyboard key is pressed and released.

Form element events:

JavaScript HTML DOM

How to add a new element to HTML DOM

Introduction.

Conclusion.

The end.

JavaScript HTML DOM

How to add a new element to HTML DOM

Introduction.

To begin with...

Conclusion.

- submit – when the visitor submits a <form>.
- focus – when the visitor focuses on an element, e.g. on an <input>.

Document events:

- DOMContentLoaded – when the HTML is loaded and processed, DOM is fully built.

CSS events:

- transitionend – when a CSS-animation finishes.

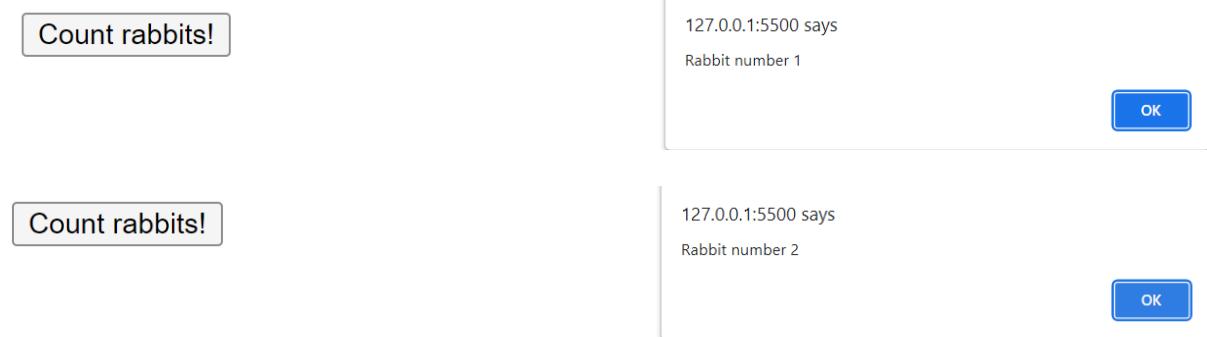
Event handlers: To react on events we can assign a handler – a function that runs in case of an event. Handlers are a way to run JavaScript code in case of user actions. A handler can be set in HTML with an attribute named on<event>. For instance, to assign a click handler for an input, we can use onclick, like here:
<input value="Click me" onclick="alert('Click!')" type="button">
On mouse click, the code inside onclick runs.

Please note that inside onclick we use single quotes, because the attribute itself is in double quotes.

Example Code & Output:

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert("Rabbit number " + i);
    }
  }
</script>

<input type="button" onclick="countRabbits()" value="Count
rabbits!">
```



DOM property

We can assign a handler using a DOM property on<event>. For instance, elem.onclick:

```
<input id="elem" type="button" value="Click me">
<script>
  elem.onclick = function() {
    alert('Thank you');
  };
</script>
```



```

<input id="elem" type="button" value="Click me">
<script>
  elem.onclick = function() {
    alert('Thank you');
  };
</script>

```



To remove a handler – assign elem.onclick = null.

Accessing the element: this

The value of this inside a handler is the element. The one which has the handler on it.

In the code below button shows its contents using this.innerHTML:

```
<button onclick="alert(this.innerHTML)">Click me</button>
```

```
<button onclick="alert(this.innerHTML)">Click me</button>
```

127.0.0.1:5500 says

Click me

AddEventListener

The syntax to add a handler:

```
element.addEventListener(event, handler, [options]);
```

event

Event name, e.g. "click".

handler

The handler function.

options

An additional optional object with properties:

- once: if true, then the listener is automatically removed after it triggers.
- capture: the phase where to handle the event, to be covered later in the chapter Bubbling and capturing. For historical reasons, options can also be false/true, that's the same as {capture: false/true}.
- passive: if true, then the handler will not call preventDefault(),

Multiple calls to addEventListener allow it to add multiple handlers

Example Code & Output:

```

<!doctype html>
<body>
<input id="elem" type="button" value="Click me"/>

<script>
  function handler1() {
    alert('Thanks!');
  }

  function handler2() {
    alert('Thanks again!');
  }

  elem.onclick = () => alert("Hello");
  elem.addEventListener("click", handler1); // Thanks!
  elem.addEventListener("click", handler2); // Thanks again!
</script>
</body>

```

Click me

127.0.0.1:5500 says
Hello

OK

Click me

127.0.0.1:5500 says
Thanks!

OK

Click me

127.0.0.1:5500 says
Thanks again!

OK

EVENTS METHOD:

Javascript has events to provide a dynamic interface to a webpage. These events are hooked to elements in the Document Object Model(DOM). These events by default use bubbling propagation i.e, upwards in the DOM from children to parent. We can bind events either as inline or in an external script.

Syntax:

<HTML-element Event-Type = "Action to be performed">

JavaScript onkeyup event: This event is a keyboard event and executes instructions whenever a key is released after pressing.

Example: In this example, we will change the color by pressing UP arrow key

```

<!doctype html>
<html>

<head>
<script>
  var a=0;
  var b=0;
  var c=0;
  function changeBackground() {
    var x=document.getElementById('bg');
    x.style.backgroundColor='rgb('+a+', '+b+', '+c+')';
    a+=100;
    b+=a+50;
    c+=b+70;
    if(a>255) a=a-b;
    if(b>255) b=a;
    if(c>255) c=b;
  }
</script>
</head>

<body>
  <h4>The input box will change color when UP arrow is pressed</h4>
  <input id="bg" onkeyup="changeBackground()" placeholder="write something" style="color:#fff">
</body>

```



The input box will change color when UP arrow key is pressed

write something

The input box will change color when UP arrow key is pressed

W

onmouseover event: This event corresponds to hovering the mouse pointer over the element and its children, to which it is bound to.

Example: In this example, we will make the box vanish when the mouse will be hovered on it

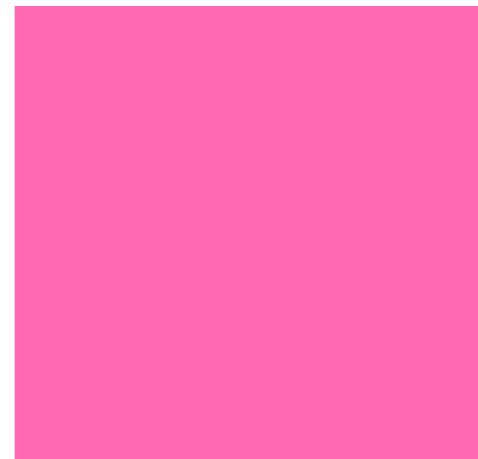
```

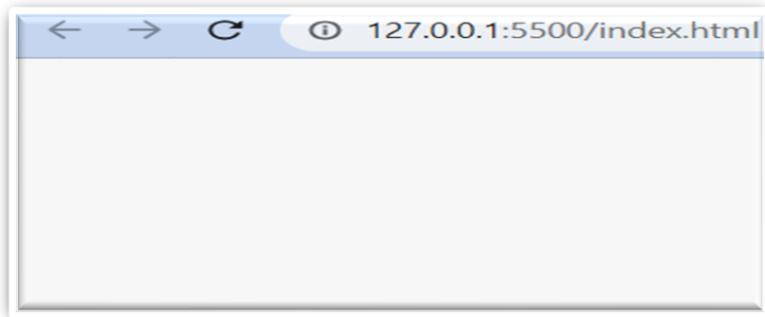
<!doctype html>
<html>

<head>
<script>
  function hov() {
    var e=document.getElementById('hover');
    e.style.display='none';
  }
</script>
</head>

<body>
  <div id="hover"
    onmouseover="hov()"
    style="background-color:#hotpink;
      height:200px;
      width:200px;">
  </div>
</body>

```





JavaScript onchange event: This event detects the change in value of any element listing to this event.

Example:

```
<!doctype html>
<html>

<head></head>

<body>
  <input onchange="alert(this.value)"
    type="number"
    style="margin-left: 45%;">
</body>

</html>
```

127.0.0.1:5500 says

1

JavaScript onfocus event: An element listing to this event executes instructions whenever it receives focus.

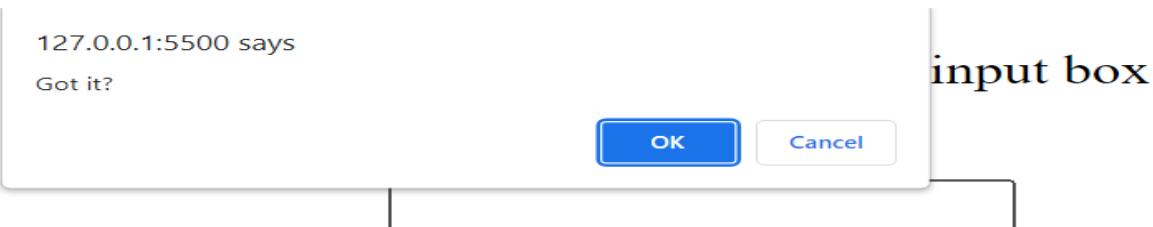
```
<!doctype html>
<!doctype html>
<html>

<head>
  <script>
    function focused() {
      var e=document.getElementById('inp');
      if(confirm('Got it?')) {
        e.blur();
      }
    }
  </script>
</head>

<body>
  <p style="margin-left: 45%;">
    Take the focus into the input box below:
  </p>
  <input id="inp"
    onfocus="focused()"
    style=" margin-left: 45%;">
</body>

</html>
```

Take the focus into the input box below:



JavaScript Animations

JavaScript animations can handle things that CSS can't.

For instance, moving along a complex path, with a timing function different from Bezier curves, or an animation on a canvas.

Using setInterval

An animation can be implemented as a sequence of frames – usually small changes to HTML/CSS properties.

For instance, changing style.left from 0px to 100px moves the element. And if we increase it in setInterval, changing by 2px with a tiny delay, like 50 times per second, then it looks smooth. That's the same principle as in the cinema: 24 frames per second is enough to make it look smooth.

The pseudo-code can look like this:

```
let timer = setInterval(function() {
  if (animation complete) clearInterval(timer);
  else increase style.left by 2px
}, 20); // change by 2px every 20ms, about 50 frames per second
```

More complete example of the animation:

```
let start = Date.now(); // remember start time
```

```
let timer = setInterval(function() {
  // how much time passed from the start?
  let timePassed = Date.now() - start;

  if (timePassed >= 2000) {
    clearInterval(timer); // finish the animation after 2 seconds
    return;
  }

  // draw the animation at the moment timePassed
  draw(timePassed);

}, 20);
```

```
// as timePassed goes from 0 to 2000
// left gets values from 0px to 400px
function draw(timePassed) {
  train.style.left = timePassed / 5 + 'px';
}
```

Example:

```

<!DOCTYPE HTML>
<html>
<head>
<style>
  #train {
    position: relative;
    cursor: pointer;
  }
</style>
</head>
<body>

<script>
  train.onclick = function() {
    let start = Date.now();

    let timer = setInterval(function() {
      let timePassed = Date.now() - start;

      train.style.left = timePassed / 5 + 'px';

      if (timePassed > 2000) clearInterval(timer);
    }, 20);
  }
</script>
</body>
</html>

```

← → C ① 127.0.0.1:5500/index.html



Let's animate the element width from 0 to 100% using our function.

```

x.html > ⚙ html
<head>
  <meta charset="utf-8">
  <style>
    progress {
      width: 5%;
    }
  </style>
  <script src="animate.js"></script>
</head>

<body>

  <progress id="elem"></progress>

  <script>
    elem.onclick = function() {
      animate({
        duration: 1000,
        timing: function(timeFraction) {
          return timeFraction;
        },
        draw: function(progress) {
          elem.style.width = progress * 100 + '%';
        }
      });
    }
  </script>

```



Timing functions

Reversal: ease*

we have a collection of timing functions. Their direct application is called “easeln”. Sometimes we need to show the animation in the reverse order. That's done with the “easeOut” transform.

easeOut

In the “easeOut” mode the timing function is put into a wrapper timingEaseOut:
timingEaseOut(timeFraction) = 1 - timing(1 - timeFraction)

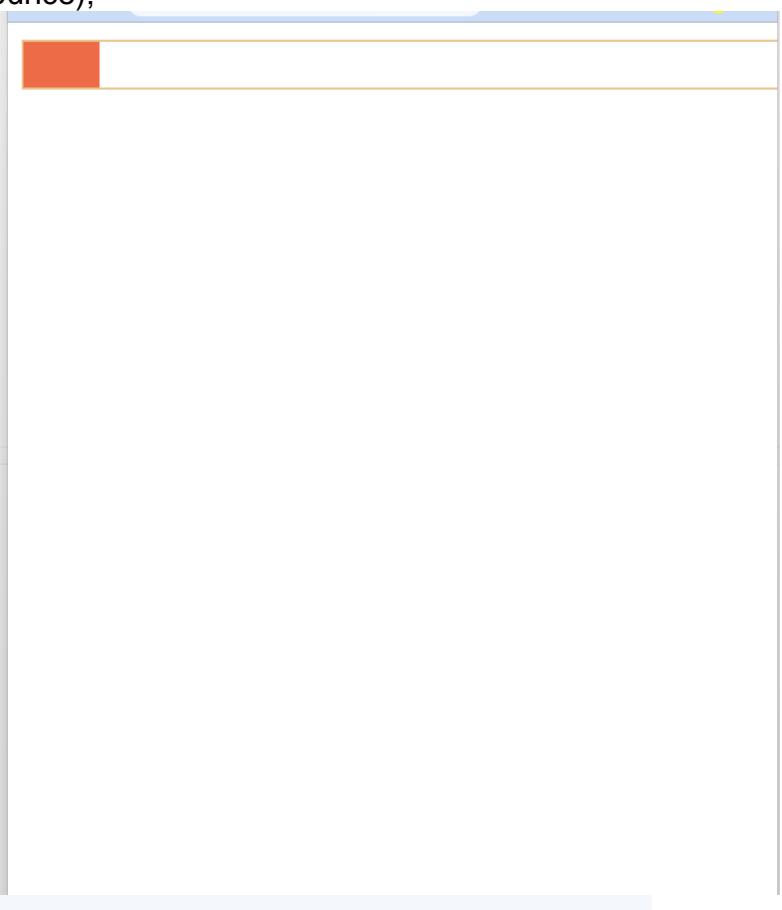
In other words, we have a “transform” function makeEaseOut that takes a “regular” timing function and returns the wrapper around it:

```
// accepts a timing function, returns the transformed variant
function makeEaseOut(timing) {
  return function(timeFraction) {
    return 1 - timing(1 - timeFraction);
  }
}
```

For instance, we can take the bounce function described above and apply it:
let bounceEaseOut = makeEaseOut(bounce);

```
<!DOCTYPE HTML>
<html>

<head>
<meta charset="utf-8">
<script src="https://js.cx/libs/animate.js"></script>
<style>
  #brick {
    width: 40px;
    height: 20px;
    background: #EE6B47;
    position: relative;
    cursor: pointer;
  }
  #path {
    outline: 1px solid #EBC48E;
    width: 540px;
    height: 20px;
  }
</style>
</head>
<body>
<div id="path">
<div id="brick"></div>
</div>
<script>
  function makeEaseOut(timing) {
    return function(timeFraction) {
      return 1 - timing(1 - timeFraction);
    }
  }
  function bounce(timeFraction) {
    for (let a = 0, b = 1; l; a += b, b /= 2) {
      if (timeFraction >= (7 - 4 * a) / 11) {
        return -Math.pow((11 - 6 * a - 11 * timeFraction) / 4, 2) + Math.pow(b, 2)
      }
    }
  }
  let bounceEaseOut = makeEaseOut(bounce);
  brick.onclick = function() {
    animate({
      duration: 3000,
      timing: bounceEaseOut,
      draw: function(progress) {
        brick.style.left = progress * 500 + 'px';
      }
    });
  };
</script>
</body>
</html>
```



Instead of moving the element we can do something else. All we need is to write the proper draw.

Here's the animated “bouncing” text typing:

The screenshot shows a web browser window with the URL 127.0.0.1:5500/index.html. On the left, the HTML source code is displayed:

```

<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8">
    <script src="https://js.cx/libs/animate.js"></script>
    <style>
        .textarea {
            display: block;
            border: 1px solid #888;
            color: #444;
            font-size: 110%;
        }
        button {
            margin-top: 10px;
        }
    </style>
</head>
<body>
    <textarea id="textExample" rows="5" cols="60">He took his vorpal sword in hand:  
Long time the manxome foe he sought—  
So rested he by the Tumtum tree,  
And stood awhile in thought.  
    </textarea>
    <button onclick="animateText(textExample)">Run the animated typing!</button>
<script>
    function animateText(textArea) {
        let text = textArea.value;
        let to = text.length;
        let from = 0;

        animate({
            duration: 5000,
            timing: bounce,
            draw: function(progress) {
                let result = (to - from) * progress + from;
                textArea.value = text.slice(0, Math.ceil(result));
            }
        });
    }

    function bounce(timeFraction) {
        for (let a = 0, b = 1; a += b, b /= 2) {
            if (timeFraction >= (7 - 4 * a) / 11) {
                return -(Math.pow((11 - 6 * a - 11 * timeFraction) / 4, 2) + Math.pow(b, 2))
            }
        }
    }
</script>
</body>
</html>

```

On the right, the output area displays the text "He took". Below it is a button labeled "Run the animated typing!". When the button is clicked, the text begins to type out letter by letter with a bouncing effect.

Cookies

What are Cookies?

A cookie is a piece of data that is stored on your computer to be accessed by your browser. You also might have enjoyed the benefits of cookies knowingly or unknowingly. Have you ever saved your Facebook password so that you do not have to type it each and every time you try to login? If yes, then you are using cookies. Cookies are saved as key/value pairs.

Why do you need a Cookie?

The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

Javascript Set Cookie

You can create cookies using document.cookie property like this.
`document.cookie = "cookiename=cookievalue"`

You can even add expiry date to your cookie so that the particular cookie will be removed from the computer on the specified date. The expiry date should be set in the UTC/GMT format. If you do not set the expiry date, the cookie will be removed when the user closes the browser.

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC"
```

You can also set the domain and path to specify to which domain and to which directories in the specific domain the cookie belongs to. By default, a cookie belongs to the page that sets the cookie.

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC; path=/ "
```

//create a cookie with a domain to the current page and path to the entire domain.

JavaScript get Cookie

You can access the cookie like this which will return all the cookies saved for the current domain.

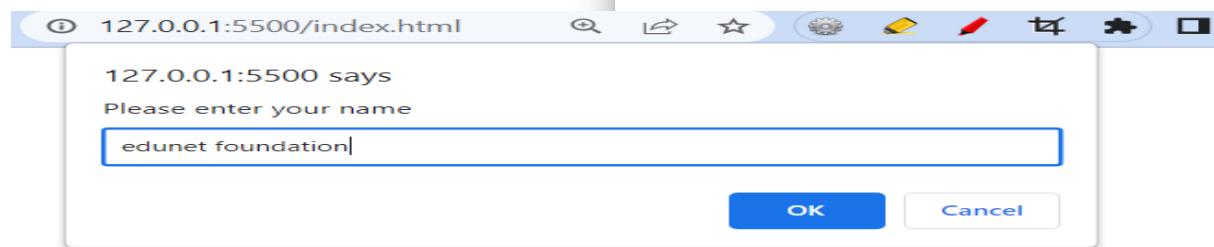
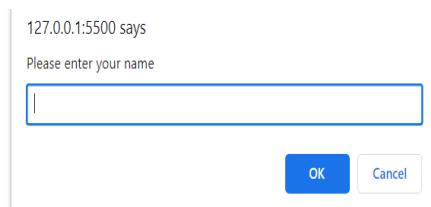
```
var x = document.cookie
```

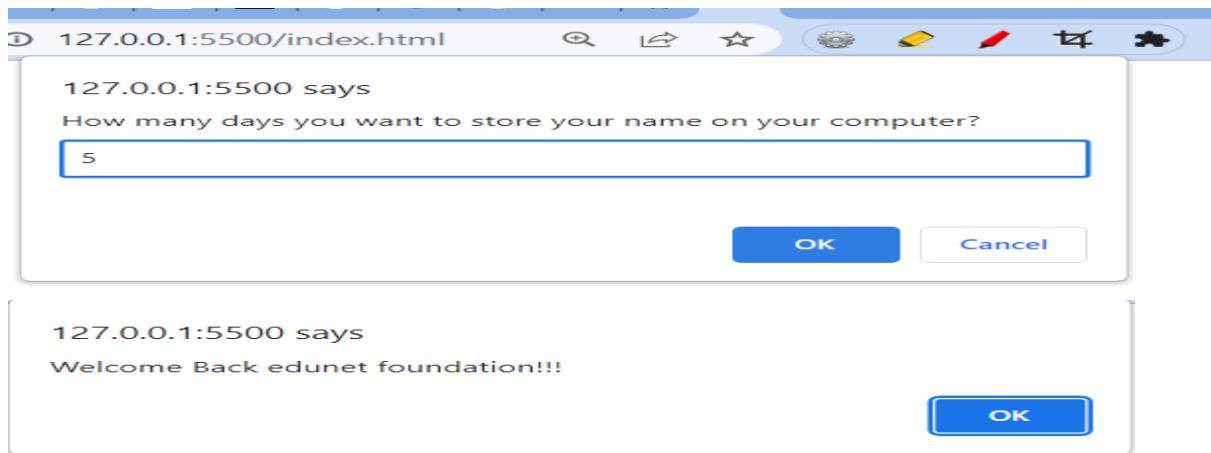
JavaScript Delete Cookie

To delete a cookie, you just need to set the value of the cookie to empty and set the value of expires to a passed date.

```
document.cookie = "cookiename= ; expires = Thu, 01 Jan 1970 00:00:00 GMT"
```

```
<html>
<head>
<title>Cookie!!!</title>
<script type="text/javascript">
function createCookie(cookieName,cookieValue,daysToExpire)
{
    var date = new Date();
    date.setTime(date.getTime()+(daysToExpire*24*60*60*1000));
    document.cookie = cookieName + "=" + cookieValue + "; expires=" + date.toGMTString();
}
function accessCookie(cookieName)
{
    var name = cookieName + "=";
    var allCookieArray = document.cookie.split(';');
    for(var i=0; i<allCookieArray.length; i++)
    {
        var temp = allCookieArray[i].trim();
        if (temp.indexOf(name)==0)
            return temp.substring(name.length,temp.length);
    }
    return "";
}
function checkCookie()
{
    var user = accessCookie("testCookie");
    if (user!="")
        alert("Welcome Back " + user + "!!!!");
    else
    {
        user = prompt("Please enter your name");
        num = prompt("How many days you want to store your name on your computer?");
        if (user!="" && user!=null)
        {
            createCookie("testCookie", user, num);
        }
    }
}
</script>
</head>
<body onload="checkCookie()"></body>
</html>
```





Session

The `sessionStorage` object stores data only for a session. It means that the data stored in the `sessionStorage` will be deleted when the browser is closed.

A page session lasts as long as the web browser is open and survives over the page refresh. When you open a page in a new tab or window, the web browser creates a new session. If you open multiple tabs or windows with the same URL, the web browser creates a separate `sessionStorage` for each tab or window. So data stored in one web browser tab cannot be accessible in another tab. When you close a tab or window, the web browser ends the session and clears data in the `sessionStorage`. Data stored in the `sessionStorage` is specific to the protocol of the page. For example, the same site `javascripttutorial.net` has different `sessionStorage` when accessing with the `http` and `https`.

Since the `sessionStorage` data is tied to a server session, it's only available when a page is requested from a server. The `sessionStorage` isn't available when the page runs locally without a server. Because the `sessionStorage` is an instance of the `Storage` type, you can manage data using the `Storage`'s methods:

- `setItem(name, value)` – set the value for a name
- `removeItem(name)` – remove the name-value pair identified by name.
- `getItem(name)` – get the value for a given name.
- `key(index)` – get the name of the value in the given numeric position.
- `clear()` – remove all values in the `sessionStorage` .

Managing data in the JavaScript `sessionStorage`

1) Accessing the `sessionStorage`

To access the `sessionStorage`, you use the `sessionStorage` property of the `window` object:

```
window.sessionStorage
```

Since the window is the global object, you can simply access the sessionStorage like this:

```
sessionStorage
```

2) Storing data in the sessionStorage

The following stores a name-value pair in the sessionStorage:

```
sessionStorage.setItem('mode','dark');
```

If the sessionStorage has an item with the name of mode, the setItem() method will update the value for the existing item to dark. Otherwise, it'll insert a new item.

3) Getting data from the sessionStorage

To get the value of an item by name, you use the getItem() method. The following example gets the value of the item 'mode':

```
const mode = sessionStorage.getItem('mode');  
console.log(mode);
```

If there is no item with the name mode, the getItem() method will return null.

4) Removing an item by a name

To remove an item by the name, you use the removeItem() method. The following removes the item with the name of 'mode':

```
sessionStorage.removeItem('mode');
```

5) Iterating over all items

To iterate over all items stored in the sessionStorage, you follow these steps:

- Use Object.keys() to get all keys of the sessionStorage object.
- Use for...of to iterate over the keys and get the items by keys.

The following code illustrates the steps:

```
let keys = Object.keys(sessionStorage);  
for(let key of keys) {  
  console.log(` ${key}: ${sessionStorage.getItem(key)} `);  
}
```

6) Deleting all items in the sessionStorage

The data stored in the sessionStorage are automatically deleted when the web browser tab/window is closed.

In addition, you can use the clear() method to programmatically delete all data stored in the sessionStorage.

`sessionStorage.clear();` Code language: CSS (css)

Why JavaScript sessionStorage

The sessionStorage has many practical applications. And the following are the notable ones:

- The sessionStorage can be used to store the state of the user interface of the web application. Later, when the user comes back to the page, you can restore the user interface stored in the sessionStorage.
- The sessionStorage can also be used to pass data between pages instead of using the hidden input fields or URL parameters.

```
<!DOCTYPE html>
<html>
<body>

<h1>The Window Object</h1>
<h2>The sessionStorage Object</h2>

<p>A Counter:</p>
<p id="demo">0</p>

<p><button onclick="clickCounter()" type="button">Count</button></p>
<p>Click to see the counter increase.</p>
<p>Close the browser tab (or window), and try again, and the counter is reset.</p>

<script>
function clickCounter() {
  if (sessionStorage.clickcount) {
    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
  } else {
    sessionStorage.clickcount = 1;
  }
  document.getElementById("demo").innerHTML = sessionStorage.clickcount;
}
</script>

</body>
</html>
```

The Window Object

The sessionStorage Object

A Counter:

2

Click to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.

The Window Object

The sessionStorage Object

A Counter:

0

Click to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.

The ... Operator

In JavaScript, there are multiple ways in which we can assign the value of one object to another. Sometimes we do not know the exact number of arguments that are to be assigned. In this case, the triple dots are used. The triple dots are known as the spread operator, which takes an iterable(array, string, or object) and expands the iterable to individual values. The spread operator is useful as it reduces the amount of code to be written and increases its readability.

Syntax: `var nameOfVar = [...valueToSpread];`

Parameter:

- **valueToSpread:** The iterable that is to be assigned to the new variable is mentioned here. It can be an Array or a String.

Return Type: It returns the argument list of the object passed after three dots.

Let us see some examples where we assign value to objects first by a simple method, then we use triple dots or spread syntax to perform the same operations.

Example 1: In this example, we will concatenate two arrays using the inbuilt concat method and then perform the same task using triple dots syntax.

```
<!Doctype html>
<html>
| <body>
| | <script>
var baseArr = [1, 2, 3];
var baseArr2 = [4, 5, 6];
// Inbuilt concat method
baseArr = baseArr.concat(baseArr2);
console.log(baseArr);
var spreadArr = ['a', 'b', 'c'];
var spreadArr2 = ['d', 'e', 'f'];
// Concatenating using three dots
spreadArr = [...spreadArr, ...spreadArr2];
console.log(spreadArr);
</script>
</body>
</html>
```

Example 2: In this example, we will copy the contents of an array with the assignment operator and the spread operator:

```
index.html > html
1  <!Doctype html>
2  <html>
3  | <body>
4  | | <script>
5  var baseArr = [1, 2, 3];
6  var baseArr2 = baseArr;
7  baseArr2.push(4);
8  console.log(baseArr2);
9  console.log(baseArr);
10 var spreadArr = ['a', 'b', 'c'];
11 var spreadArr2 = [...spreadArr];
12 spreadArr2.push('d');
13 console.log(spreadArr);
14 console.log(spreadArr2);
15 </script>
16 </body>
17 </html>
18
```

Explanation: We can see that when assigning the array using the assignment operator('=') and modifying the new array, the old array is also modified. This causes a problem as we do not always want to modify the content of the old array but if we use spread operator syntax to assign values and modify the new array, the old array is left unchanged. This happens because a new array is returned by the spread operator instead of the reference of the old array.

Example 3: In this example, we will find the min of an array using the Math.min() method:

```
<!Doctype html>
<html>
| <body>
| | <script>
var baseArr = [5, 2, 7, 8, 4, 9];
console.log(Math.min(baseArr));
console.log(Math.min(...baseArr));
|</script>
|</body>
|</html>
```

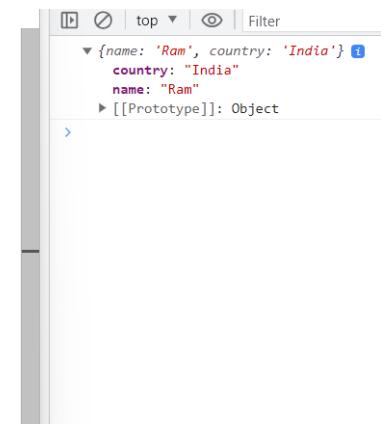


Explanation: The **math.min()** method requires an object list to find the minimum inside the list but when passing an array it gives NaN output. To overcome this problem we used the triple dots format/spread operator. As the spread operator returns a list of objects it is accepted by the method and the minimum element of the array is returned.

Example 4: In this example, we will assign an object to another object using the three dots:

ix.html > html > body

```
<!Doctype html>
<html>
| <body>
| | <script>
const spreadObj = {
  name: 'Ram',
  country: 'India',
};
// Cloning previous object
const newObj = { ...spreadObj };
console.log(newObj);
</script>
|</body>
|</html>
```



Example 5: In this example, we will use the spread operator in a function call

```
x.html / ↗ num / ↗ body / ↗ script
<!Doctype html>
<html>
  <body>
    <script>
function add(...objects){
  var ans = 0;
  for(var i = 0; i < objects.length; i++){
    ans += objects[i];
  }
  console.log(ans);
}
add(1, 2);
add(23, 45, 67, 56);
</script>
</body>
</html>
```



Explanation: We can see that the spread operator is useful when we do not know the number of arguments that are to be passed in a function. Here our add function works for any number of arguments so we do not have to specify multiple functions for a different number of arguments.

The rest parameter is an improved way to handle function parameters, allowing us to more easily handle various inputs as parameters in a function. The rest parameter syntax allows us to represent an indefinite number of arguments as an array. With the help of a rest parameter, a function can be called with any number of arguments, no matter how it was defined. Rest parameter is added in ES2015 or ES6 which improved the ability to handle parameter.

Syntax:

```
//... is the rest parameter (triple dots)
function functionname(...parameters)
{
  statement;
}
```

Note: When ... is at the end of the function parameter, it is the rest parameter. It stores n number of parameters as an array. Let's see how the rest parameter works:

Example: Let's see how the rest parameter works:

```

<!Doctype html>
<html>
  <body>
    <script>
      // Without rest parameter
      function fun(a, b){
        return a + b;
      }
      console.log(fun(1, 2)); // 3
      console.log(fun(1, 2, 3, 4, 5)); // 3
    </script>
  </body>
</html>

```

In the above code, no error will be thrown even when we are passing arguments more than the parameters, but only the first two arguments will be evaluated. It's different in the case of the rest parameter. With the use of the rest parameter, we can gather any number of arguments into an array and do what we want with them.

Example: Javascript code demonstrating the addition of numbers using the rest parameter.

```

<!Doctype html>
<html>
  <body>
    <script>
      // es6 rest parameter
      function fun(...input){
        let sum = 0;
        for(let i of input){
          sum+=i;
        }
        return sum;
      }
      console.log(fun(1,2)); //3
      console.log(fun(1,2,3)); //6
      console.log(fun(1,2,3,4,5)); //15
    </script>
  </body>
</html>

```

Let's make use of the rest parameter with some other arguments inside a function:

```

<!Doctype html>
<html>
  <body>
    <script>
      // rest with function and other arguments
      function fun(a,b,...c){
        console.log(` ${a} ${b}`); //Mukul Latiyan
        console.log(c); // [ 'Lionel', 'Messi', 'Barcelona' ]
        console.log(c[0]); //Lionel
        console.log(c.length); //3
        console.log(c.indexOf('Lionel')); //0
      }
      fun('Mukul','Latiyan','Lionel','Messi','Barcelona');
    </script>
  </body>
</html>

```

```

Mukul Latiyan
▼ (3) [ 'Lionel', 'Messi', 'Barcelona' ] ⓘ
  0: "Lionel"
  1: "Messi"
  2: "Barcelona"
  length: 3
▶ [[Prototype]]: Array(0)
Lionel
3
0
>

```

In the above code sample, we passed the rest parameter as the third parameter, and then we basically called the function `fun()` with five arguments the first two were treated normally and the rest were all collected by the rest parameter hence we get 'Lionel' when we tried to access `c[0]` and it is also important to note that the rest parameter gives an array in return

JavaScript for... of Loop

The syntax of the for...of loop is:

```

for (element of iterable) {
  // body of for...of
}

```

Here,

- iterable - an iterable object (array, set, strings, etc).
- element - items in the iterable

for...of with Arrays

The for..of loop can be used to iterate over an array. For example,

```

<!Doctype html>
<html>
  <body>
    <script>
      // array
      const students = ['John', 'Sara', 'Jack'];
      // using for...of
      for ( let element of students ) {
        // display the values
        console.log(element);
      }
    </script>
  </body>
</html>

```

```

John
Sara
Jack
Live reload enabled.
>

```

In the above program, the for...of loop is used to iterate over the `students` array object and display all its values.

for...of with Strings

You can use for...of loop to iterate over string values. For example,

```

<!Doctype html>
<html>
|   <body>
|   |   <script>
|   |   // string
|   |   const string = 'code';
|
|   // using for...of loop
|   for (let i of string) {
|       console.log(i);
|   }
|   </script>
|   </body>
|> </html>

```

```

c
o
d
e

```

for...of with Sets

You can iterate through Set elements using the for...of loop. For example,

```

<!Doctype html>
<html>
|   <body>
|   |   <script>
|   |   // define Set
|   |   const set = new Set([1, 2, 3]);
|
|   // looping through Set
|   for (let i of set) {
|       console.log(i);
|   }
|   </script>
|   </body>
|> </html>

```

```

1
2
3

```

for...of with Maps

You can iterate through Map elements using the for...of loop. For example,

```

<!Doctype html>
<html>
|   <body>
|   |   <script>
|   |   // define Map
|   |   let map = new Map();
|
|   // inserting elements
|   map.set('name', 'Jack');
|   map.set('age', '27');
|
|   // looping through Map
|   for (let [key, value] of map) {
|       console.log(key + ' - ' + value);
|   }
|   </script>
|   </body>
|> </html>

```

```

name - Jack
age - 27

```

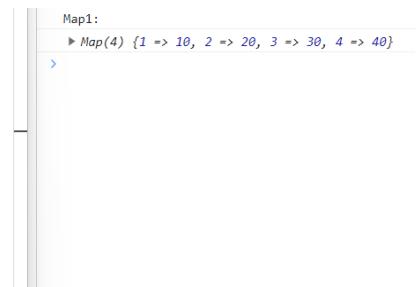
JS Map

The map is a collection of elements where each element is stored as a **Key, value** pair. Map objects can hold both objects and primitive values as either key or

value. When we iterate over the map object it returns the key, and value pair in the same order as inserted. [Map\(\) constructor](#) is used to create Map in JavaScript.

```
<!DOCTYPE html>
<body>
  <script>
let map1 = new Map([
  [1 , 10], [2 , 20] ,
  [3, 30],[4, 40]
]);

console.log("Map1: ");
console.log(map1);
  </script>
</body>
</html>
```



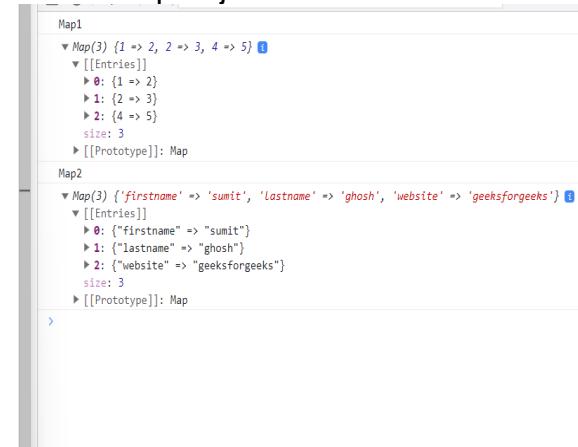
Map object provided by **ES6**. A key of a Map may occur once, which will be unique in the map's collection. There are slight advantages to using a map rather than an object.

- **Accidental Keys & Security:** No default keys are stored, only contain what's explicitly put into them. Because of that, it's safe to use.
- **Key Types & Order:** It can be any value as a key function, object anything. And the order is straightforward way in the order of entry insertion.
- **Size:** Because of the size property a map can be easily retrieved.
- **Performance:** Any operation can be performed on map so easily in a better way.
- **Serialization and parsing:** We can create our own serialization and parsing support for Map by using `JSON.stringify()` and `JSON.parse()` methods.

Example 1: In this example, we will create a basic map object

```
<!DOCTYPE html>
<body>
  <script>
let map1 = new Map([
  [1, 2],
  [2, 3],
  [4, 5]
]);
console.log("Map1");
console.log(map1);
let map2 = new Map([
  ["firstname", "sumit"],
  ["lastname", "ghosh"],
  ["website", "geeksforgeeks"]
]);
console.log("Map2");
console.log(map2);

  </script>
</body>
</html>
```

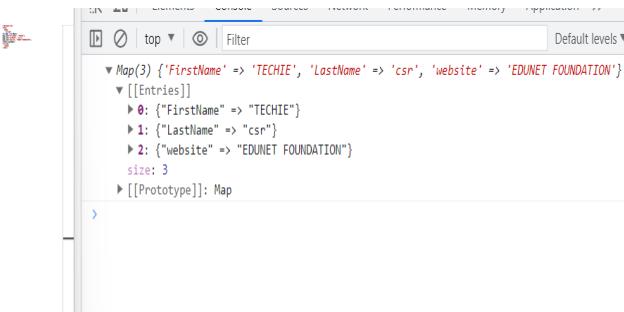


Example 2: This example adds elements to the map using [set\(\)](#) method.

```

ex.html > ...
<!DOCTYPE html>
<html>
  <body>
    <script>
      let map1 = new Map();
      map1.set("FirstName", "TECHIE");
      map1.set("LastName", "csr");
      map1.set("website", "EDUNET FOUNDATION");
      console.log(map1);
    </script>
  </body>
</html>

```

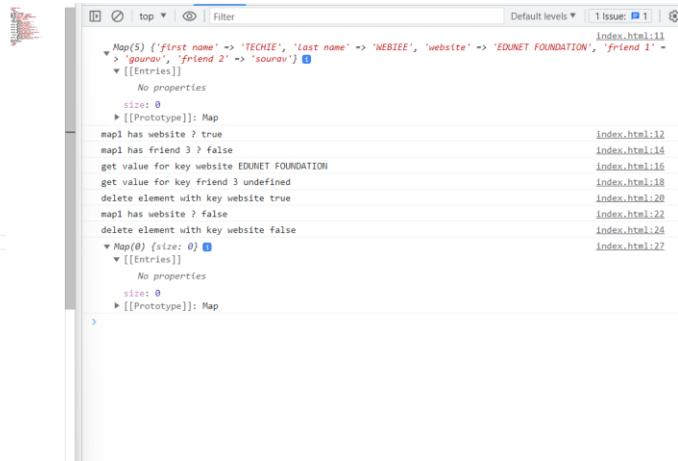


Example 3: This example explains the use of Map methods like has(), get(), delete(), and clear().

```

<!DOCTYPE html>
<html>
  <body>
    <script>
      let map1 = new Map();
      map1.set("First name", "TECHIE");
      map1.set("last name", "WEBIEE");
      map1.set("website", "EDUNET FOUNDATION")
        .set("friend 1", "gourav")
        .set("friend 2", "sourav");
      console.log(map1);
      console.log("map1 has website ? " +
        map1.has("website"));
      console.log("map1 has friend 3 ? " +
        map1.has("friend 3"));
      console.log("get value for key website " +
        map1.get("website"));
      console.log("get value for key friend 1 " +
        map1.get("friend 1"));
      console.log("delete element with key website " +
        + map1.delete("website"));
      console.log("map1 has website ? " +
        map1.has("website"));
      console.log("delete element with key website " +
        + map1.delete("friend 3"));
      map1.clear();
      console.log(map1);
    </script>
  </body>
</html>

```



JS SET

A set is a collection of items that are unique i.e no element can be repeated. Set in ES6 are ordered: elements of the set can be iterated in the insertion order. Set can store any type of value whether primitive or objects.

Syntax:

```
new Set([it]);
```

Parameter:

- it: It is an iterable object whose all elements are added to the new set created, If the parameter is not specified or null is passed then a new set created is empty.

Returns: A new set object

CODE:

```

// ["sumit","amit","anil","anish"]
let set1 = new Set(["sumit", "sumit", "amit", "anil", "anish"]);

// it contains 'f', 'o', 'd'
let set2 = new Set("foooooooood");

// it contains [10, 20, 30, 40]
let set3 = new Set([10, 20, 30, 30, 40, 40]);

```

```
// it is an empty set
let set4 = new Set();
Properties:
Set.size – It returns the number of elements in the Set.
Methods of Set:
Set.add() – It adds the new element with a specified value at the end of the Set object.
Syntax:
set1.add(val);
Parameter:
• val: It is a value to be added to the set.
Return value: The set object
```

```
<!Doctype html>
<html>
  <body>
    <script>
let set1 = new Set();

set1.add(10);
set1.add(20);

// As this method returns
// the set object hence chaining
// of add method can be done.
set1.add(30).add(40).add(50);

console.log(set1);

</script>
</body>
</html>
```

Set(5) {10, 20, 30, 40, 50} [Entries]
 ▶ 0: 10
 ▶ 1: 20
 ▶ 2: 30
 ▶ 3: 40
 ▶ 4: 50
 size: 5
 [[Prototype]]: Set

Set.delete() – It deletes an element with the specified value from the Set object.

Syntax:
set1.delete(val);

Parameter:
• val: It is a value to be deleted from the set.

Return value: true if the value is successfully deleted from the set else returns false.

```
<body>
  <script>
let set1 = new Set("foooodiiieee");

// deleting e from the set
// it prints true
console.log(set1.delete('e'));

console.log(set1);

// deleting an element which is
// not in the set
// prints false
console.log(set1.delete('g'));

</script>
</body>
</html>
```

true
 Set(4) {'f', 'o', 'd', 'i'} [Entries]
 ▶ 0: "f"
 ▶ 1: "o"
 ▶ 2: "d"
 ▶ 3: "i"
 size: 4
 [[Prototype]]: Set
 false

Set.clear() – It removes all the element from the set.

Syntax:

```
set1.clear();
```

Parameter: This method does not take any parameter

Return value: Undefined

The screenshot shows a browser's developer tools console. On the left, there is a code editor window containing the following JavaScript code:

```
<body>
  <script>
let set1 = new Set("foooodiiieee");
let set2 = new Set([10, 20, 30, 40, 50]);

console.log(set2);

set2.clear()

console.log(set2);
  </script>
</body>
</html>
```

On the right, the browser's developer tools pane shows the state of the variables. It shows two sets: `set1` and `set2`. `set1` contains the string "foooodiiieee". `set2` initially contains the numbers [10, 20, 30, 40, 50]. After the `set2.clear()` call, `set2` is empty, as indicated by the size being 0 and the log output showing an empty set.

Set.forEach(): It executes the given *function* once for every element in the Set, in the insertion order.

Syntax:

```
set1.forEach(callback[,thisargument]);
```

Parameter:

- **callback** – It is a function that is to be executed for each element of the Set.
 - The callback function is provided with three parameters as follows:
 - the *element key*
 - the *element value*
 - the *Set object* to be traversed
- **thisargument** – Value to be used as this when executing the callback.

Return value: Undefined

Set.prototype[@@iterator](): It returns a Set iterator function which is *values()* function by default.

Syntax:

```
set1[Symbol.iterator]();
```

Parameter: This method does not take any parameter

Return value: A Set iterator function and it is *values()* by default.

Example:

```

<body>
  <script>
    let set1 = new Set(["sumit","sumit","amit","anish"]);
    let getit = set1[Symbol.iterator]();
    console.log(getit.next());
    console.log(getit.next());
    console.log(getit.next());
    console.log(getit.next());
  </script>
</body>
</html>

```

The screenshot shows a browser's developer tools with the "Elements" tab open. On the left, there is a code editor window displaying the provided JavaScript code. On the right, the "Elements" panel shows the DOM structure of the page. It lists a single script element. Inside the script element, there is a Set object with four entries: 'sumit', 'sumit', 'amit', and 'anish'. Each entry is represented as an object with properties 'value' and 'done'. The 'value' property contains the string value, and the 'done' property is set to false. There are also [[Prototype]] entries for each object.

JavaScript Promise and Promise Chaining

In JavaScript, a promise is a good way to handle asynchronous operations. It is used to find out if the asynchronous operation is successfully completed or not.

A promise may have one of three states.

- Pending
- Fulfilled
- Rejected

A promise starts in a pending state. That means the process is not complete. If the operation is successful, the process ends in a fulfilled state. And, if an error occurs, the process ends in a rejected state.

For example, when you request data from the server by using a promise, it will be in a pending state. When the data arrives successfully, it will be in a fulfilled state. If an error occurs, then it will be in a rejected state.

Create a Promise

To create a promise object, we use the `Promise()` constructor.

```
let promise = new Promise(function(resolve, reject){
  //do something
});
```

The `Promise()` constructor takes a function as an argument. The function also accepts two functions `resolve()` and `reject()`.

If the promise returns successfully, the `resolve()` function is called. And, if an error occurs, the `reject()` function is called.

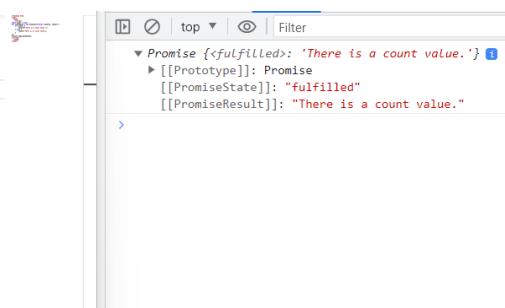
Let's suppose that the program below is an asynchronous program. Then the program can be handled by using a promise.

Example 1: Program with a Promise

```

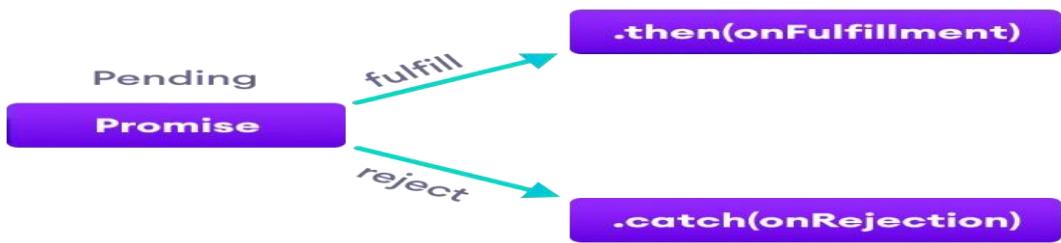
<body>
  <script>
    const count = true;
    let countValue = new Promise(function (resolve, reject) {
      if (count) {
        resolve("There is a count value.");
      } else {
        reject("There is no count value");
      }
    });
    console.log(countValue);
  </script>
</body>
</html>

```



In the above program, a Promise object is created that takes two functions: resolve() and reject(). resolve() is used if the process is successful and reject() is used when an error occurs in the promise.

The promise is resolved if the value of count is true.



JavaScript Promise Chaining

Promises are useful when you have to handle more than one asynchronous task, one after another. For that, we use promise chaining.

You can perform an operation after a promise is resolved using methods then(), catch() and finally().

JavaScript then() method

The then() method is used with the callback when the promise is successfully fulfilled or resolved.

The syntax of then() method is:

```
promiseObject.then(onFulfilled, onRejected);
```

Example 2: Chaining the Promise with then()

The screenshot shows a browser's developer tools console window. At the top, there are icons for search, refresh, and filter, followed by the text "Promise resolved". Below that, a message says "You can call multiple functions this way." with a small arrow pointing to the right. The main area of the console shows the following code and its execution results:

```
<body>
  <script>

// returns a promise

let countValue = new Promise(function (resolve, reject) {
|  resolve("Promise resolved");
});

// executes when promise is resolved successfully

countValue
  .then(function successValue(result) {
|  console.log(result);
})

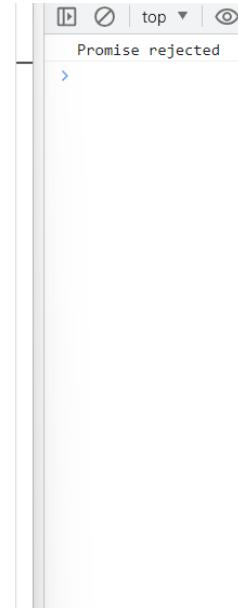
  .then(function successValue1() {
|  console.log("You can call multiple functions this way.");
});|>
</script>
| </body>
</html>
```

In the above program, the `then()` method is used to chain the functions to the promise. The `then()` method is called when the promise is resolved successfully. You can chain multiple `then()` methods with the promise.

JavaScript catch() method

The `catch()` method is used with the callback when the promise is rejected or if an error occurs.

For example,

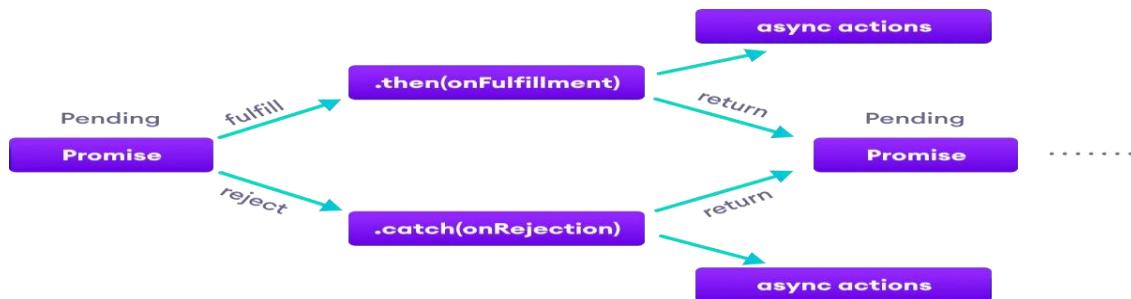


```
<body>
  <script>
// returns a promise
let countValue = new Promise(function (resolve, reject) {
  | | reject('Promise rejected');
});

// executes when promise is resolved successfully
countValue.then(
  | | function successValue(result) {
    | | | | console.log(result);
  },
)

// executes if there is an error
.catch(
  | | function errorValue(result) {
    | | | | console.log(result);
}
);
</script>
</body>
</html>
```

In the above program, the promise is rejected. And the `catch()` method is used with a promise to handle the error.



```

<body>
  <script>
// returns a promise
let countValue = new Promise(function (resolve, reject) {
  // could be resolved or rejected
  resolve('Promise resolved');
});

// add other blocks of code
countValue.finally(
  function greet() {
    | | console.log('This code is executed.');
  }
);
</script>
</body>
</html>

```

JavaScript Promise Versus Callback

Promises are similar to callback functions in a sense that they both can be used to handle asynchronous tasks. JavaScript callback functions can also be used to perform synchronous tasks.

Their differences can be summarized in the following points:

Java Script Promise	Java Script CallBack
The syntax is user-friendly and easy to read	The syntax is difficult to understand
Error handling is easier to manage.	Error handling may be hard to manage
Example: <pre>api().then(function(result) { return api2(); }).then(function(result2) { return api3(); }).then(function(result3) { // do work }).catch(function(error) { // handle any error that may occur before this point });</pre>	Example: <pre>api(function(result){ api2(function(result2){ api3(function(result3){ // do work if(error) { // do something } else { // do something } }); }); });</pre>

JavaScript finally() method

You can also use the `finally()` method with promises. The `finally()` method gets executed when the promise is either resolved successfully or rejected.

Modules

As our application grows bigger, we want to split it into multiple files, so called “modules”. A module may contain a class or a library of functions for a specific purpose. For a long time, JavaScript existed without a language-level module syntax. That wasn’t a problem, because initially scripts were small and simple, so there was no need. But eventually scripts became more and more complex, so the community invented a variety of ways to organize code into modules, special libraries to load modules on demand.

What is a module?

A module is just a file. One script is one module. As simple as that. Modules can load each other and use special directives export and import to interchange functionality, call functions of one module from another one:

- export keyword labels variables and functions that should be accessible from outside the current module.
 - import allows the import of functionality from other modules.

For instance, if we have a file sayHi.js exporting a function:

```
// 📁 sayHi.js  
  
export function sayHi(user) {  
  alert(`Hello, ${user}!`);  
}
```

...Then another file may import and use it:

```
// 📁 main.js

import {sayHi} from './sayHi.js';

alert(sayHi); // function...

sayHi('John'); // Hello, John!
```

The import directive loads the module by path `./sayHi.js` relative to the current file, and assigns exported function `sayHi` to the corresponding variable.

Let's run the example in-browser.

As modules support special keywords and features, we must tell the browser that a script should be treated as a module, by using the attribute `<script type="module">`.

Like this:

```
EXPLORER ... Welcome index.html JS say.js ...
MMY
index.html
say.js

JS say.js > sayHi
1 export function sayHi(user) {
2   return `Hello, ${user}!`;
3 }

index.html > {} "index.html" > script
1 <!doctype html>
2 <script type="module">
3 import {sayHi} from './say.js';
4
5 document.body.innerHTML = sayHi('John');
6 </script>
```

Output:



Hello, John!

The browser automatically fetches and evaluates the imported module (and its imports if needed), and then runs the script.

Core module features

There are core features, valid both for browser and server-side JavaScript.

Always “use strict”

Modules always work in strict mode. E.g. assigning to an undeclared variable will give an error.

```
<script type="module">
  a = 5; // error
</script>
```

Module-level scope

Each module has its own top-level scope. In other words, top-level variables and functions from a module are not seen in other scripts.

In the example below, two scripts are imported, and `hello.js` tries to use `user` variable declared in `user.js`. It fails, because it's a separate module (you'll see the error in the console):

The image shows a code editor interface with three tabs open:

- `index.html`:
```html<!DOCTYPE html><script type="module" src="user.js"></script><script type="module" src="hello.js"></script>```
- `hello.js`:  
```jsalert(user); // no such variable (each module has independent variables)```
- `user.js`:
```jslet user = "John";```

The `EXPLORER` sidebar on the right shows the project structure with files `hello.js`, `index.html`, and `user.js`.

Modules should export what they want to be accessible from outside and import what they need.

- `user.js` should export the `user` variable.
- `hello.js` should import it from `user.js` module.

In other words, with modules we use `import/export` instead of relying on global variables.

This is the correct variant:

```

<!doctype html>
<script type="module" src="hello.js"></script>

hello.js
import {user} from './user.js';
document.body.innerHTML = user; // John

user.js
export let user = "John";

```

← → C 127.0.0.1:5500

# John

## What is jQuery?

jQuery is a lightweight Javascript library which is blazing fast and concise. This library was created by John Resig in 2006 and jQuery has been designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax. jQuery can be used to find a particular HTML element in the HTML document with a certain ID, class or attribute and later we can use jQuery to change one or more of attributes of the same element like color, visibility etc. jQuery can also be used to make a webpage interactive by responding to an event like a mouse click.

jQuery has been developed with the following principles:

- Separation of JavaScript and HTML, which encourages developers to completely separate JavaScript code from HTML markup.
- Brevity and clarity promotes features like chainable functions and shorthand function names.
- Eliminates of cross-browser incompatibilities, so developers does not need to worry about browser compatibility while writing code using jQuery library.
- Extensibility, which means new events, elements, and methods can be easily added in jQuery library and then reused as a plugin.

## jQuery Features

jQuery simplifies various tasks of a programmer by writing less code. Here is the list of important core features supported by jQuery –

- DOM manipulation – The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called Sizzle.
- Event handling – The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- AJAX Support – The jQuery helps you a lot to develop a responsive and featurerich site using AJAX technology.
- Animations – The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- Lightweight – The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).

- Cross Browser Support – The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- Latest Technology – The jQuery supports CSS3 selectors and basic XPath syntax.

## Setting up jQuery

There are two ways to use jQuery.

1. Local Installation – You can download jQuery library on your local machine and include it in your HTML code.
2. CDN Based Installation – You can include jQuery library into your HTML code directly from Content Delivery Network (CDN).

### jQuery - Local Installation

You can download latest version of jQuery on your web server and include the downloaded library in your code. We suggest you to download compressed version of the library for a better performance.

- Go to the <https://jquery.com/download/> to download the latest version available.
- Now put downloaded jquery-3.6.0.min.js file in a directory of your website, e.g. /jquery/jquery-3.6.0.js.
- Finally include this file in your HTML markup file as shown below.

### jQuery - CDN Based Installation

You can include jQuery library into your HTML code directly from a Content Delivery Network (CDN). There are various CDNs which provide a direct link to the latest jQuery library which you can include in your program.

## Example

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
 $(document).ready(function() {
 document.write("Hello, World!");
 });
</script>
</head>
<body>
 <!-- HTML body goes here -->
</body>
</html>
<html>
```

Hello, World!

## JQuery Selectors

jQuery Selectors are used to select HTML element(s) from an HTML document. Consider an HTML document is given and you need to select all the <div> from this document. This is where jQuery Selectors will help.

jQuery Selectors can find HTML elements (ie. Select HTML elements) based on the following:

- HTML element Name
- Element ID

- Element Class
- Element attribute name
- Element attribute value
- Many more criteria

The jQuery library harnesses the power of Cascading Style Sheets (CSS) selectors to let us quickly and easily access elements or groups of elements in the Document Object Model (DOM).

jQuery Selectors works in very similar way on an HTML document like an SQL Select Statement works on a Database to select the records.

### jQuery Selector Syntax

Following is the jQuery Selector Syntax for selecting HTML elements:

```
$(document).ready(function(){
 $(selector)
});
```

A jQuery selector starts with a dollar sign \$ and then we put a selector inside the braces (). Here \$() is called factory function, which makes use of following three building blocks while selecting elements in a given document:

#### The element Selector

The jQuery element selector selects HTML element(s) based on the element name. Following is a simple syntax of an element selector:

```
$(document).ready(function(){
 $("Html Element Name")
});
```

Please note while using element name as jQuery Selector, we are not giving angle braces alongwith the element. For example, we are giving only plain p instead of <p>.

## Examples

Following is an example to select all the <p> elements from an HTML document and then change the background color of those paragraphs. You will not see any <p> element in the output generated by this example.

```
<!DOCTYPE html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
 $(document).ready(function() {
 $("p").css("background-color", "yellow");
 });
</script>
</head>
<body>
 <h1>jQuery element Selector</h1>

 <p>This is p tag</p>
 This is span tag
 <div>This is div tag</div>
</body>
</html>
```

## jQuery element Selector

This is p tag

This is span tag  
This is div tag

## The #id Selector

The jQuery #id selector selects an HTML element based on the element id attribute. Following is a simple syntax of a #id selector:

```
$(document).ready(function(){
 $("#id of the element")
});
```

To use jQuery #id selector, you need to make sure that id attribute should be uniquely assigned to all the DOM elements. If your elements will have similar ids then it will not produce correct result.

Examples:

Following is an example to select the <p> element whose id is foo and change the background color of those paragraphs.

```
<!DOCTYPE html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
 $(document).ready(function() {
 $("#foo").css("background-color", "yellow");
 });
</script>
</head>
<body>
 <h1>jQuery #id Selector</h1>

 <p id="foo">This is foo p tag</p>
 This is bar span tag
 <div id="bill">This is bill div tag</div>
</body>
</html>
```

## jQuery #id Selector

This is foo p tag

This is bar span tag

This is bill div tag

## The .class Selector

The jQuery .class selector selects HTML element(s) based on the element class attribute. Following is a simple syntax of a .class selector:

```
$(document).ready(function(){
 $(".class of the element")
});
```

Because a class can be assigned to multiple HTML elements within an HTML document, so it is very much possible to find out multiple elements with a single .class selector statement.

Examples:

Following is an example to select all the elements whose class is foo and change the background color of those elements.

```
<!DOCTYPE html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
 $(document).ready(function() {
 $(".foo").css("background-color", "yellow");
 });
</script>
</head>
<body>
 <h1>jQuery .class Selector</h1>

 <p class="foo">This is foo p tag</p>
 <p class="foo">This is one more foo p tag</p>
 This is bar span tag
 <div class="bill">This is bill div tag</div>
</body>
</html>
```

## jQuery .class Selector

This is foo p tag

This is one more foo p tag

This is bar span tag

This is bill div tag

## What are jQuery Events?

A jQuery Event is the result of an action that can be detected by jQuery (JavaScript). When these events are triggered, you can then use a custom function to do pretty much whatever you want with the event. These custom functions are called Event Handlers.

### Example:

The screenshot shows a browser window with the following code in the head section:

```
<head>
 <title>The jquery Example</title>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
 <script>
 $(document).ready(function() {
 $("div").click(function(){
 alert('Hi there!');
 });
 });
 </script>
 <style>
 div{ margin:10px;padding:12px; border:2px solid #666; width:60px;cursor:pointer}
 </style>
</head>
<body>
 <p>Click on any of the squares to see the result:</p>

 <div>One</div>
 <div>Two</div>
 <div>Three</div>
</body>
</html>
```

The body contains three div elements with the text "One", "Two", and "Three" respectively. A tooltip appears over the "One" button with the text "Click on any of the squares to see the result:". When the "One" button is clicked, a modal dialog box appears with the message "127.0.0.1:5500 says Hi there!" and an "OK" button.

## JQUERY DOM MANIPULATION

jQuery provides methods such as attr(), html(), text() and val() which act as getters and setters to manipulate the content from HTML documents.

### jQuery - Get Content

jQuery provides html() and text() methods to extract the content of the matched HTML element.

Following is the syntax of these two methods:

```
$(selector).html();
```

```
$(selector).text();
```

The jQuery text() method returns plain text value of the content where as html() returns the content with HTML tags. You will need to use jQuery selectors to select the target element.

## EXAMPLE

```

<!doctype html>
<html>
<head>
<title>The jquery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function() {
 $("#text").click(function(){
 alert($("#p").text());
 });
 $("#html").click(function(){
 alert($("#p").html());
 });
});
</script>
</head>
<body>

The quick brown fox jumps over the lazy dog</p>

<button id="text">Get Text</button>
<button id="html">Get HTML</button>
</body>
</html>


```

The quick **brown fox** jumps over the **lazy dog**

[Get Text](#) [Get HTML](#)

The quick

[Get Text](#)

OK

127.0.0.1:5500 says

The quick brown fox jumps over the lazy dog

The quick

[Get Text](#)

OK

127.0.0.1:5500 says

The quick <b>brown fox</b> jumps over the <b>lazy dog</b>

## jQuery – Effects

jQuery effects add an X factor to your website interactivity. jQuery provides a trivially simple interface for doing various kind of amazing effects like show, hide, fade-in, fade-out, slide-up, slide-down, toggle etc. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

### Example

#### Hiding Elements

jQuery gives simple syntax to hide an element with the help of `hide()` method:

`$(selector).hide( [speed, callback] );`

You can apply any jQuery selector to select any DOM element and then apply jQuery `hide()` method to hide it. Here is the description of all the parameters which gives you a solid control over the hiding effect –

- speed – This optional parameter represents one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- callback – This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

The default speed duration 'normal' is 400 milliseconds. The strings 'fast' and 'slow' can be supplied to indicate durations of 200 and 600 milliseconds, respectively. Higher values indicate slower animations.

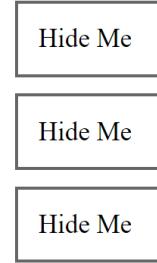
### Example

Following is an example where a <div> will hide itself when we click over it. We have used 1000 as speed parameter which means it will take 1 second to apply the hide effect on the clicked element.

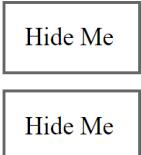
```
<!doctype html>
<html>
<head>
<title>The jquery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function() {
 $("div").click(function(){
 $(this).hide(1000);
 });
});
</script>
<style>
div{ margin:10px;padding:12px; border:2px solid #666; width:60px; cursor:pointer}
</style>
</head>
<body>
<p>Click on any of the squares to see the result:</p>

<div>Hide Me</div>
<div>Hide Me</div>
<div>Hide Me</div>
</body>
</html>
```

Click on any of the squares to see the result:



Click on any of the squares to see the result:



## Click on any of the squares to see the result:

### jQuery - Animation

The jQuery animate() method is used to create custom animations by changing the CSS numerical properties of a DOM element, for example, width, height, margin, padding, opacity, top, left, etc.

Following is a simple syntax of animate() method:

```
$(selector).animate({ properties }, [speed, callback]);
```

jQuery animate() method can not be used to animate non-numeric properties such as color or background-color etc. Though you can use jQuery plugin jQuery.Color to animate such properties.

You can apply any jQuery selector to select any DOM element and then apply jQuery animate() method to animate it. Here is the description of all the parameters which gives you a complete control over the animation –

- properties – A required parameter which defines the CSS properties to be animated and this is the only mandatory parameter of the call.

- speed – An optional string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- callback – An optional parameter which represents a function to be executed whenever the animation completes.

### Animation Pre-Requisite

(a) - The animate() method does not make hidden elements visible as part of the effect. For example, given \$(selector).hide().animate({height: "20px"}, 500), the animation will run, but the element will remain hidden.

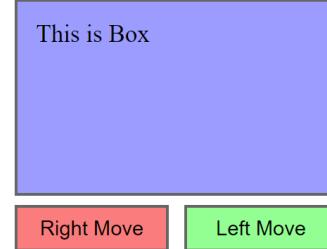
(b) - To manipulate the position of a DOM element as a part of the animation, first we need to set its position to relative, fixed, or absolute because by default, all HTML elements have a static position, and they cannot be moved using animate() method.

### Example:

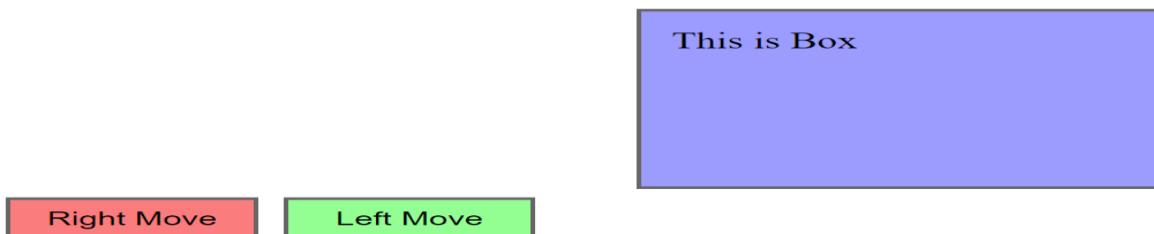
The following example shows how to use animate() method to move a <div> element to the right, until it has reached a left property of 250px. Next when we click left button, the same <div> element returns to its initial position.

```
<!DOCTYPE html>
<html>
<head>
<title>The jQuery Example</title>
<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function() {
 $("#right").click(function(){
 $("div").animate({left: '250px'});
 });
 $("#left").click(function(){
 $("div").animate({left: '0px'});
 });
});
</script>
<style>
#left, #right{margin:3px; border:2px solid #666; height:30px; width:100px; cursor:pointer;}
#box{position:relative; margin:3px; padding:12px; border:2px solid #666; height:100px; width:180px;}
</style>
</head>
<body>
 <p>Click on Left or Right button to see the result:</p>
 <div id="box" style="background-color:#99ccff;">This is Box</div>
 <button id="right" style="background-color:#fb7c7c;">Right Move</button>
 <button id="left" style="background-color:#93ff93;">Left Move</button>
</body>
</html>
```

Click on Left or Right button to see the result:



Click on Left or Right button to see the result:



Click on Left or Right button to see the result:



### jQuery - DOM Traversing

jQuery is a very powerful tool which provides a variety of DOM traversal methods to help us select elements in an HTML or XML document randomly as well as in sequential method. Elements in the DOM are organized into a tree-like data structure

that can be traversed to navigate, locate the content within an HTML or XML document.

Document Object Model (DOM) - is a W3C (World Wide Web Consortium) standard that allows us to create, change, or remove elements from the HTML or XML documents.

The DOM tree can be imagined as a collection of nodes related to each other through parent-child and sibling-sibling relationships and the root start from the top parent which is HTML element in an HTML document.

Before we start traversing a DOM, Let's understand the terminology of parent, child and sibling. Let's see an example:

```
<body>
 <p>This is paragraph</p>

 <div>This is div</div>

 <button id="b1">Get width</button>
 <button id="b2">Set width</button>
</body>
```

In the above example, we have a `<body>` element at the top, which is called a parent for all the elements. The `<div>`, `<p>` and `<button>` elements inside the `<body>` element are called siblings. Again `<span>` element inside `<div>` is a child of `<div>` and `<div>` is called a parent of `<span>` element.

The `<div>` element is a next sibling of the `<p>` element and `<p>` is the previous sibling of the `<div>` element.

## Traversing the DOM

Most of the DOM Traversal Methods do not modify the jQuery DOM object and they are used to filter out elements from a document based on given conditions. jQuery provides methods to traverse in the following three directions:

- Traversing Upwards - This direction means traversing the ancestors (Parent, Grandparent, Great-grandparent etc.)
- Traversing Downwards - This direction means traversing the descendants (Child, Grandchild, Great-grandchild etc.)
- Sideways - This direction means traversing the ancestors the siblings (Brother, sisters available at the same level)

### jQuery parent() Method

The jQuery `parent()` method returns the direct parent of the each matched element. Following is a simple syntax of the method:

```
$(selector).parent([filter])
```

Example:

```

ix.html > ⌂ html
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function() {
 $("button").click(function(){
 $(".child-two").parent().css("border", "2px solid red");
 });
});
</script>
<style>
.great-grand-par
ent *{display:block; border:2px solid #aaa; color:#aaa; padding:5px; margin:5px;}
</style>
</head>
<body>
 <div class="great-grand-parent">
 <div style="width:500px;" class="grand-parent">
 <ul class="parent">
 <li class="child-one">Child One
 <li class="child-two">Child Two

 </div>
 </div>

 <button>Mark Parent</button>
</body>
</html>

```

- Child One
- Child Two

**Mark Parent**

- Child One
- Child Two

**Mark Parent**

## JQUERY JSON

The **getJSON()** method in jQuery fetches JSON-encoded data from the server using a GET HTTP request.

### Syntax:

`$(selector).getJSON(url,data,callback)`

**Parameters:** This method accepts three parameters as mentioned above and described below:

- url: It is a required parameter. It is used to specify the URL in the form of a string to which the request is sent
- data: It is an optional parameter that specifies data that will be sent to the server.
- callback: It is also an optional parameter that runs when the request succeeds.

**Return Value:** It returns XMLHttpRequest object.

**Example:** The below example illustrates the **getJSON()** method in jQuery.  
**employee.json file:**



```

<html> </html> </body> </section> </table> </table> </script> </body>
<body>
 <h1>JQUERY JSON DATA</h1>

 <!-- TABLE CONSTRUCTION-->
 <table id="table">
 <!-- HEADING FORMATION -->
 <tr>
 <th>GFG UserHandle</th>
 <th>Practice Problems</th>
 <th>Coding Score</th>
 <th>GFG Articles</th>
 </tr>
 </table>
</body>

```

```

1) gfgdetails.json > ...
2) [{"GFGUserName": "User-1", "NoOfProblems": "150", "TotalScore": "100", "Articles": "30"}]

```

```

<script>
 $(document).ready(function () {
 // FETCHING DATA FROM JSON FILE
 $.getJSON("gfgdetails.json",
 function (data) {
 var student = '';
 // ITERATING THROUGH OBJECTS
 $.each(data, function (key, value) {
 //CONSTRUCTION OF ROWS HAVING
 // DATA FROM JSON OBJECT
 student += '<tr>';
 student += '<td>' +
 value.GFGUserName + '</td>';
 student += '<td>' +
 value.NoOfProblems + '</td>';

 student += '<td>' +
 value.TotalScore + '</td>';

 student += '<td>' +
 value.Articles + '</td>';

 student += '</tr>';
 });
 //INSERTING ROWS INTO TABLE
 $('#table').append(student);
 });
 });
</script>
</section>
</body>

```

```

</html>

```

## JQUERY JSON DATA

GFG UserHandle	Practice Problems	Coding Score	GFG Articles
User-1	150	100	30

### jQuery ajax() Method

The ajax() method in jQuery is used to perform an AJAX request or asynchronous HTTP request.

#### Syntax:

`$.ajax({name:value, name:value, ... })`

**Example 1:** This example use ajax() method to add the text content using ajax request.

The screenshot shows a code editor with two tabs open. The left tab, titled 'index.html', contains the following code:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>
6 | jQuery ajax() Method
7 </title>
8
9 <script src=
10 "https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
11 </script>
12
13 <script>
14 $(document).ready(function() {
15 $("li:parent").css("background-color", "green");
16 });
17 </script>
18 </head>
19
20 <body style="text-align:center;">
21
22 <h1 style="color:green">
23 | JQUERY AJAX METHOD
24 </h1>
25
26 <h2>
27 | jQuery ajax() Method
28 </h2>
29
30 <h3 id="h11"></h3>
31
32 <button>Click</button>
33
34 <!-- Script to use ajax() method to
35 add text content -->
36 <script>
37 $(document).ready(function() {
38 $("button").click(function() {
39 $.ajax({
40 url: "plain.txt",
41 success: function(result) {
42 $("#h11").html(result);
43 }
44 });
45 });
46 </script>
47
48 </html>
```

The right tab, titled 'plain.txt', contains the text 'WELCOME TO JQUERY AJAX METHOD'.

# JQUERY AJAX METHOD

## jQuery ajax() Method

Click

127.0.0.1:5500/index.html

# JQUERY AJAX METHOD

## jQuery ajax() Method

WELCOME TO JQUERY AJAX METHOD

Click

# Chapter 2: Essentials of Java Programming

## Learning Outcomes:

- Understand the fundamental concepts of Java programming language
- Gain a solid knowledge of object-oriented programming (OOP) concepts
- Develop robust and efficient Java applications

## 2.1 Java Installation

Check if Java Is Installed. Before installing the Java Development Kit, check if a Java version is already installed on Windows. Follow the steps below:

1. Open a command prompt by typing **cmd** in the search bar and press **Enter**.
2. Run the following command: **java -version**

## Steps to Install Java

### Step 1: Download Java for Windows 10

Download the latest Java Development Kit installation file for Windows 10 to have the latest features and bug fixes. Using your preferred web browser, navigate to the Oracle Java Downloads page.

On the Downloads page, click the **x64 Installer** download link under the **Windows** category. At the time of writing this article, Java version 17 is the latest long-term support Java version.

Linux	macOS	Windows
Product/file description	File size	Download
x64 Compressed Archive	170.66 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip</a> (sha256)
x64 Installer	152 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe</a> (sha256)
x64 MSI Installer	150.89 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi</a> (sha256)

Wait for the download to complete.

### Step 2: Install Java on Windows 10

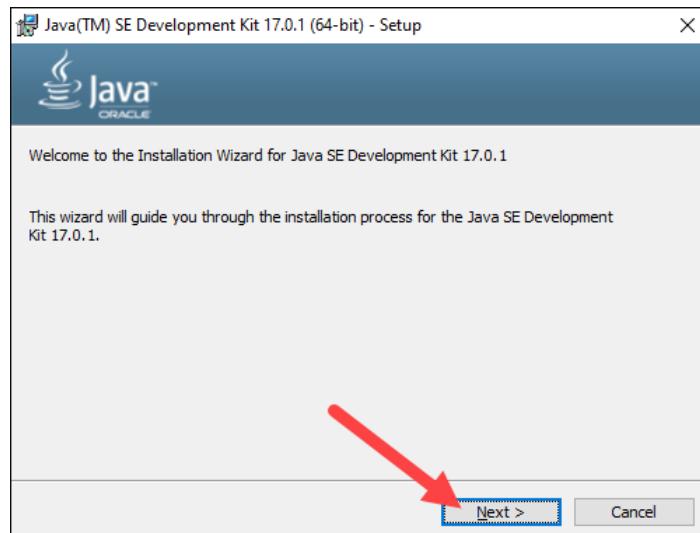
After downloading the installation file, proceed with installing Java on your Windows system.

Follow the steps below:

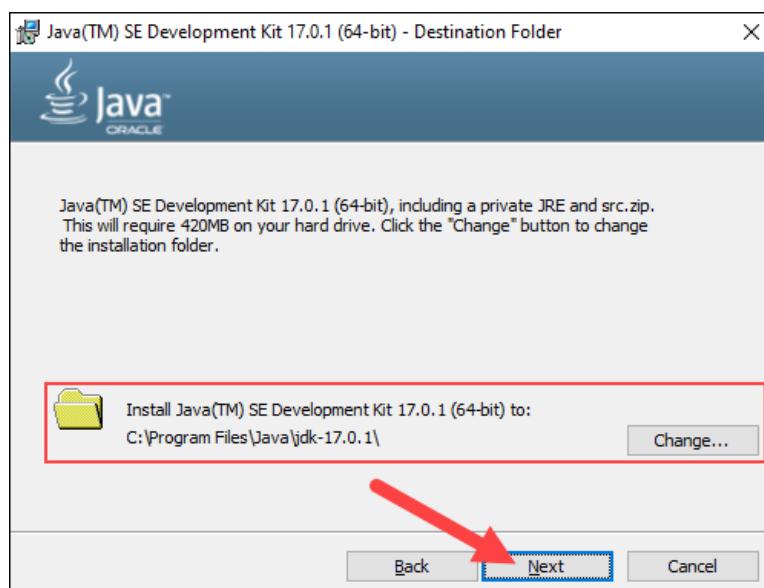
**Step 1:** Run the Downloaded File. Double-click the **downloaded file** to start the installation.

**Step 2:** Configure the Installation Wizard. After running the installation file, the installation wizard welcome screen appears.

1. Click **Next** to proceed to the next step.



2. Choose the destination folder for the Java installation files or stick to the default path. Click **Next** to proceed.



3. Wait for the wizard to finish the installation process until the Successfully Installed message appears. Click **Close** to exit the wizard.

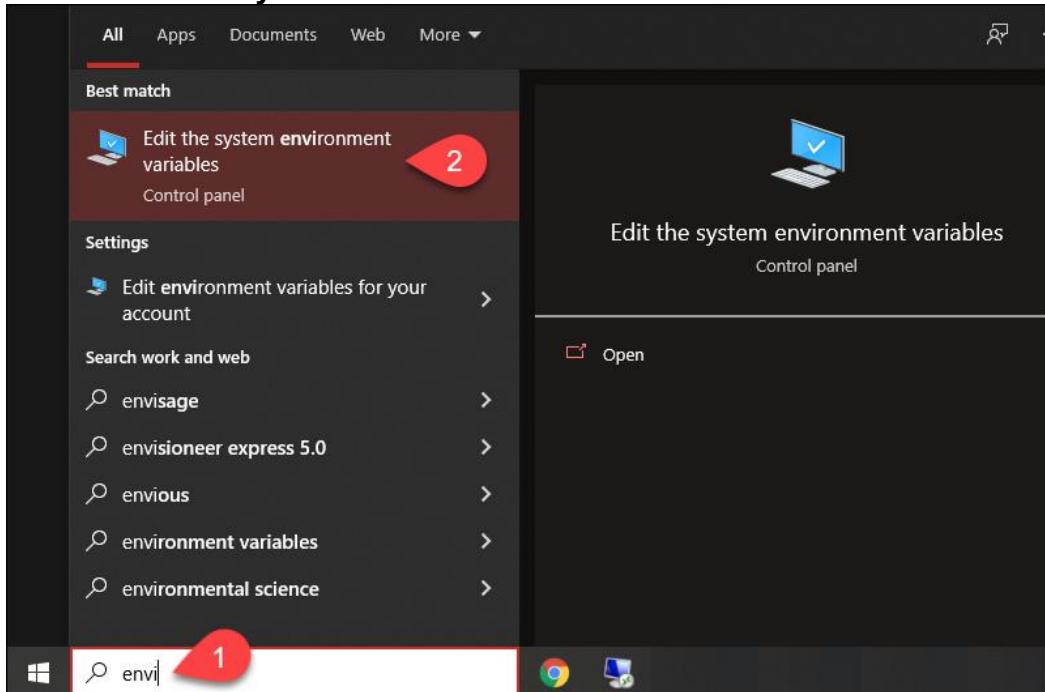


## Set Environmental Variables in Java

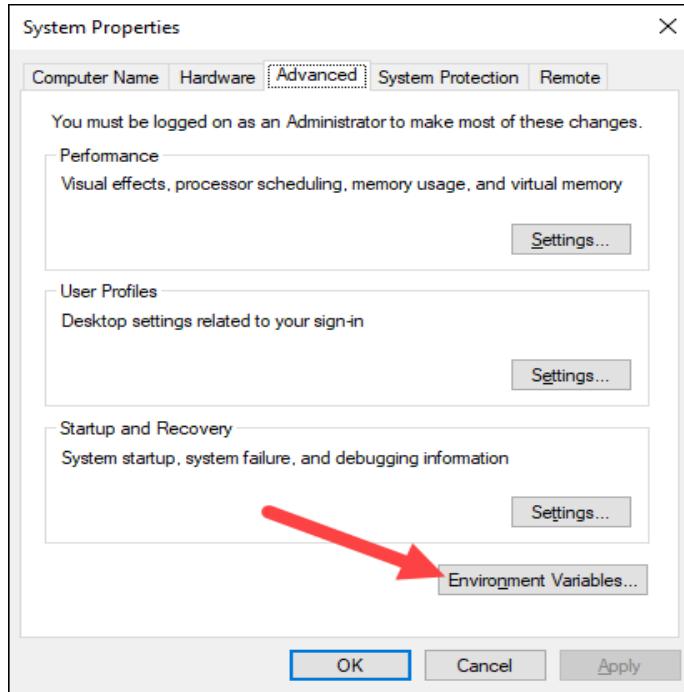
Set Java environment variables to enable program compiling from any directory. To do so, follow the steps below:

### Step 1: Add Java to System Variables

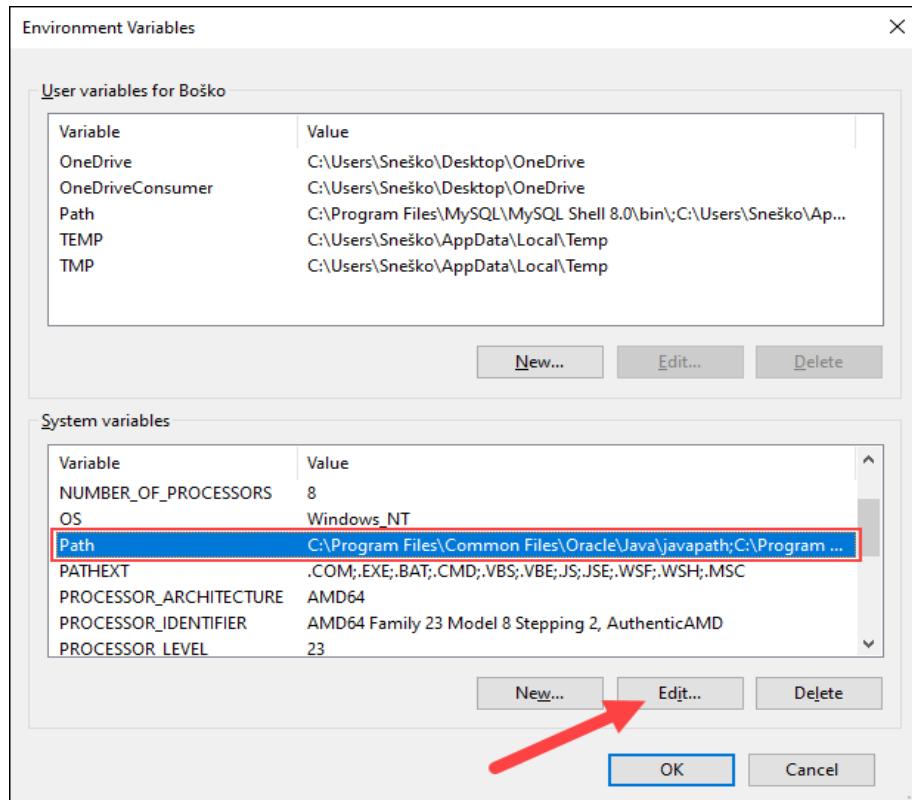
1. Open the **Start** menu and search for *environment variables*.
2. Select the **Edit the system environment variables** result.



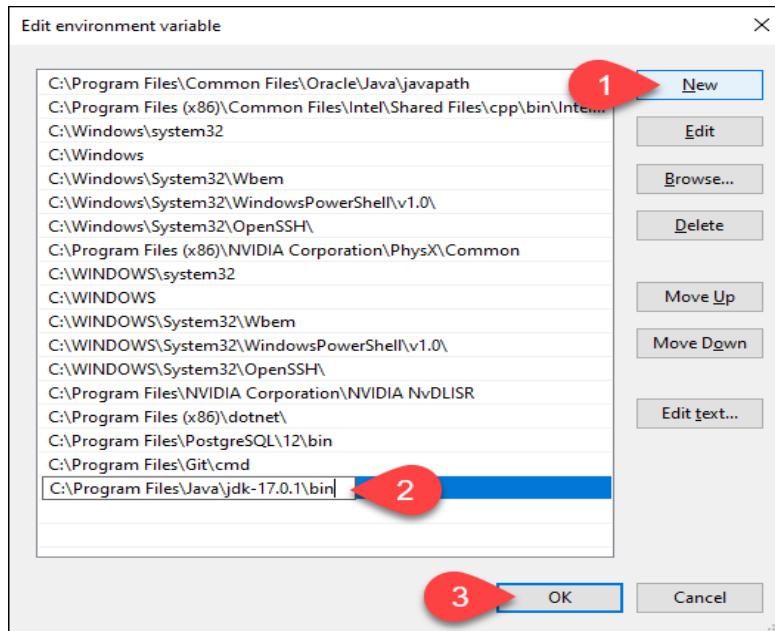
3. In the System Properties window, under the Advanced tab, click **Environment Variables...**



4. Under the System variables category, select the **Path** variable and click **Edit**:



5. Click the **New** button and enter the path to the Java bin directory:



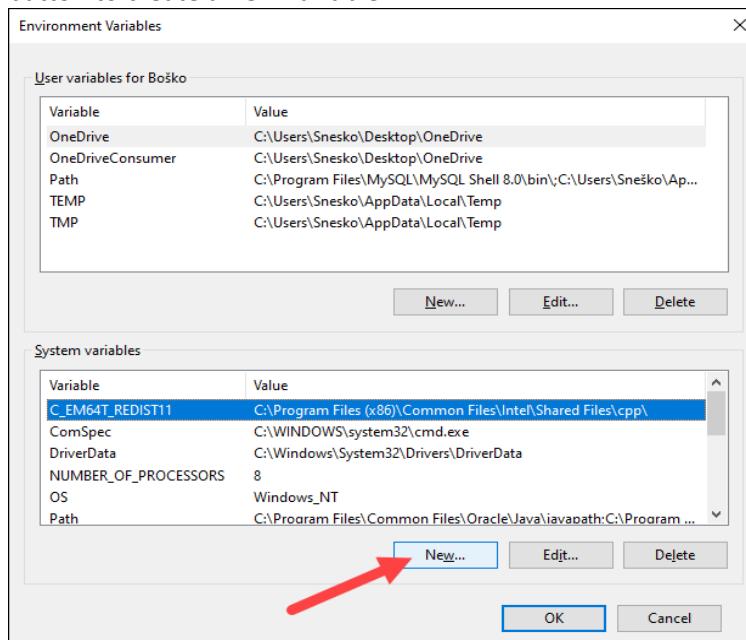
**Note:** The default path is usually **C:\Program Files\Java\jdk-17.0.1\bin**.

6. Click **OK** to save the changes and exit the variable editing window.

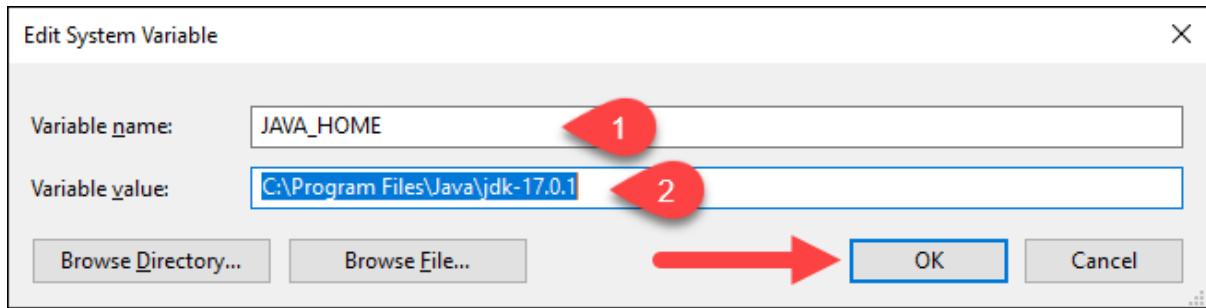
### Step 2: Add JAVA\_HOME Variable.

Some applications require the **JAVA\_HOME** variable. Follow the steps below to create the variable:

1. In the Environment Variables window, under the System variables category, click the **New...** button to create a new variable.



2. Name the variable as **JAVA\_HOME**.
3. In the variable value field, paste the path to your Java jdk directory and click **OK**.



4. Confirm the changes by clicking OK in the Environment Variables and System properties windows.

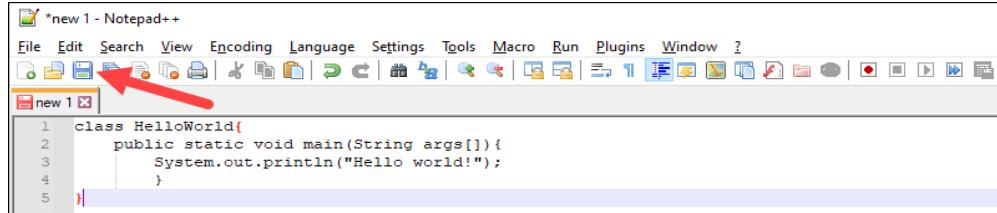
## Execution of the Program

If installed correctly, the command outputs the Java version. Make sure everything works by writing a simple program and compiling it. Follow the steps below:

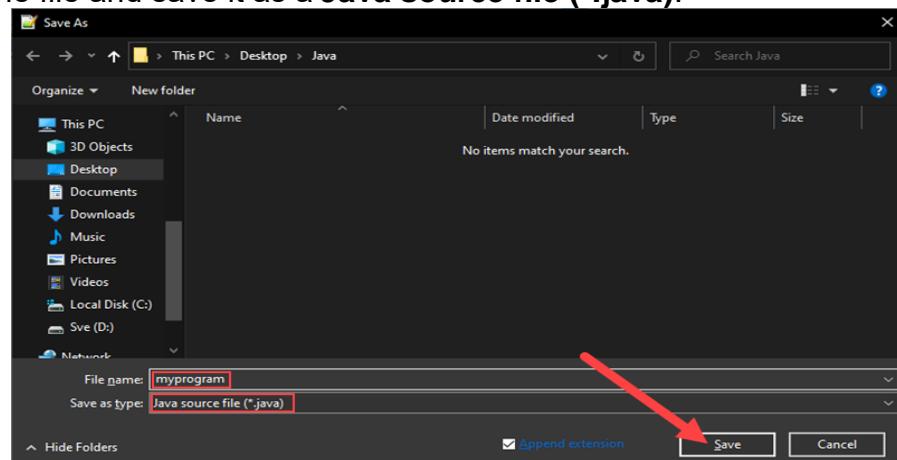
### Step 1: Write a Test Java Script

1. Open a text editor such as Notepad++ and create a new file.
2. Enter the following lines of code and click **Save**:

```
class HelloWorld{
 public static void main(String args[]){
 System.out.println("Hello world!");
 }
}
```



3. Name the file and save it as a Java source file (\*.java).



**Note:** When using Notepad, select **All files** for the Save as type option and add the .java extension to the file name.

## Step 2: Compile the Test Java Script

1. In the command prompt, change the directory to the file's location and use the following syntax to compile the program:

**Javac [filename].java**

For example:

```
C:\Users\boskom\Desktop\Java>javac myprogram.java
C:\Users\boskom\Desktop\Java>
```

After a successful compilation, the program generates a .class file in the file directory.

2. Run the program with the following syntax:

**java [filename]**

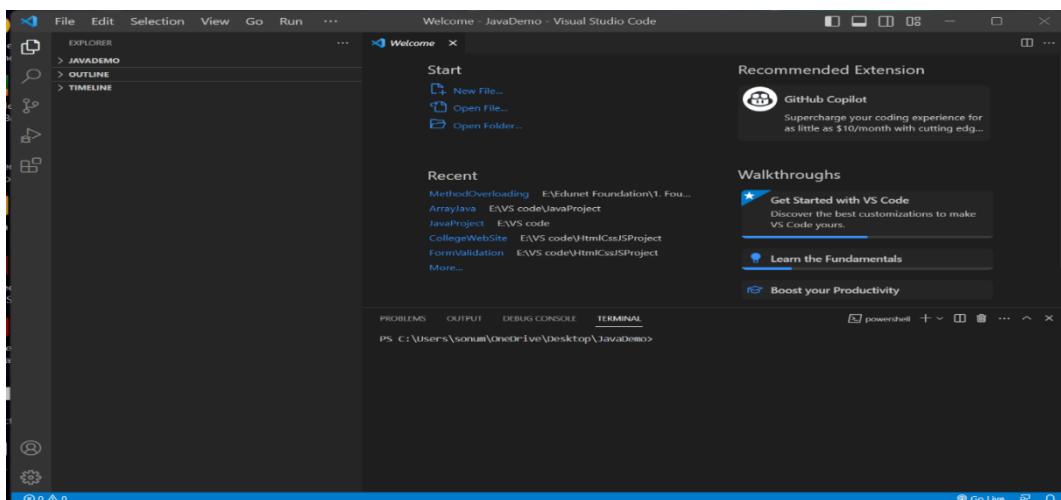
```
C:\Users\boskom\Desktop\Java>java HelloWorld
Hello world!
```

The output shows that the program runs correctly, displaying the **Hello world!** message.

## Visual Studio Code on Windows

### Installation of Java in VS Code (Another way to Run Java Code)

1. Download the [Visual Studio Code installer](#) for Windows.
2. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). This will only take a minute.
3. By default, VS Code is installed under <C:\Users\{Username}\AppData\Local\Programs\Microsoft VS Code>.



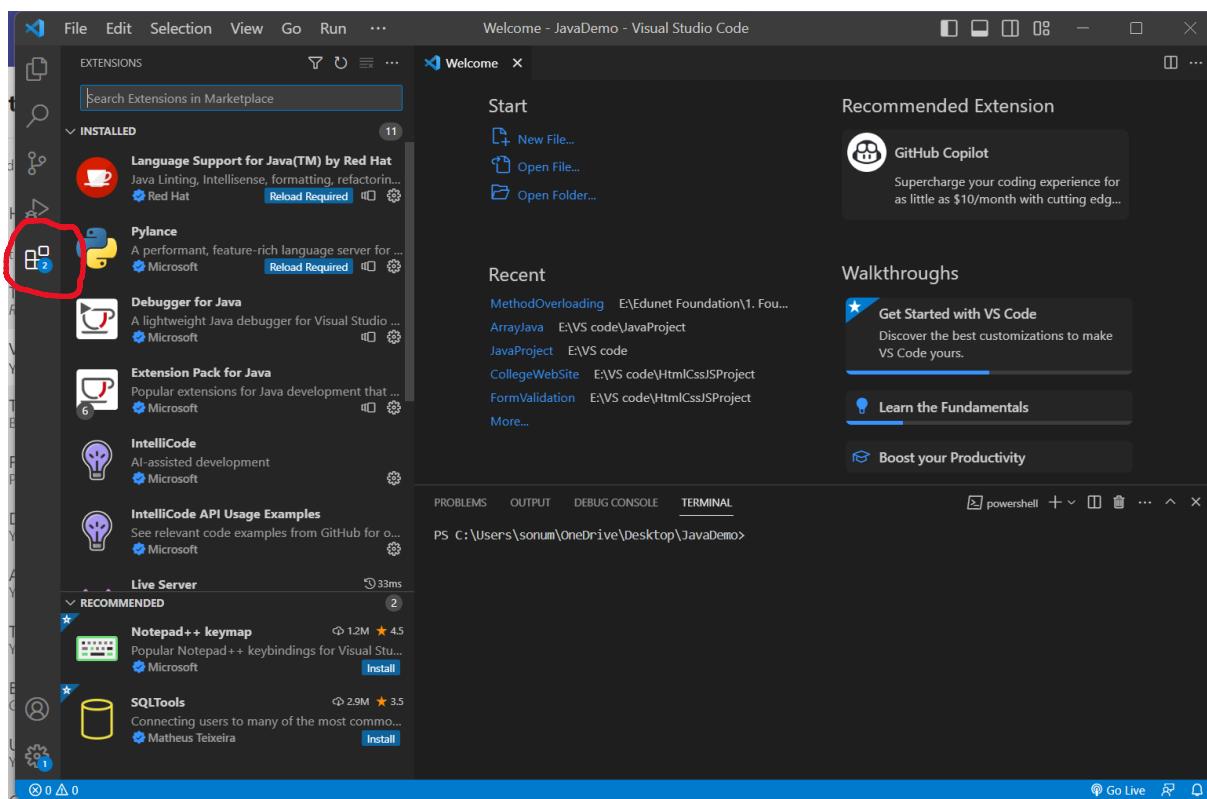
## Getting Started with Java in VS Code

To help you set up quickly, you can install the **Coding Pack for Java**, which includes VS Code, the Java Development Kit (JDK), and essential Java extensions. The

Coding Pack can be used as a clean installation, or to update or repair an existing development environment.

If you are an existing VS Code user, you can also add Java support by installing the [Extension Pack for Java](#), which includes these extensions:

- Language Support for Java™ by Red Hat
- Debugger for Java
- Test Runner for Java
- Maven for Java
- Project Manager for Java
- Visual Studio IntelliCode



## 2.2 Java Fundamentals

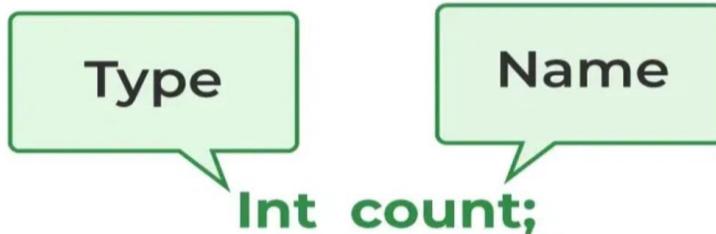
### Java Variables

In Java, Variables are the data containers that save the data values during Java program execution. Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold. A variable is a memory location name for the data. Java variable is a name given to a memory location. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution.

Variables in Java are only a name given to a memory location. All the operations done on the variable affect that memory location. In Java, all variables must be declared before use.

## How to Declare Variables in Java?

We can declare variables in Java as pictorially depicted below as a visual aid.



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

1. **datatype:** Type of data that can be stored in this variable.
2. **data\_name:** Name was given to the variable.

In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

## How to Initialize Variables in Java?

It can be perceived with the help of 3 components that are as follows:

- datatype: Type of data that can be stored in this variable.
- variable\_name: Name given to the variable.
- value: It is the initial value stored in the variable.

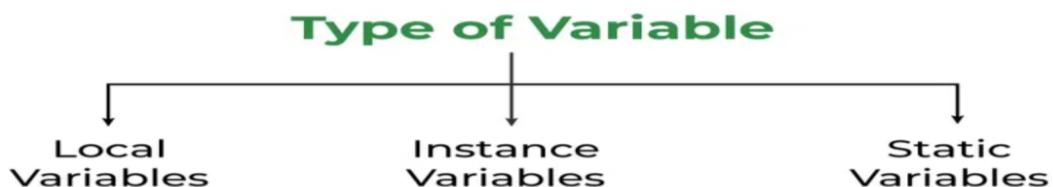


Reference: [Java Variables - GeeksforGeeks](#)

## Types of Variables in Java

Now let us discuss different types of variables which are listed as follows:

1. Local Variables
2. Instance Variables
3. Static Variables



Reference: [Java Variables - GeeksforGeeks](#)

### 1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

File Name :- GFG.java

```
// Java Program to implement
// Local Variables
import java.io.*;

class GFG {
 public static void main(String[] args)
 {
 // Declared a Local Variable
 int var = 10;

 // This variable is local to this main method only
 System.out.println("Local Variable: " + var);
 }
}
```

Output:

```
Local Variable: 10
```

## 2. Instance Variables

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of an instance variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for Wrapper classes like *Integer* it is *null*, etc.
- Instance variables can be accessed only by creating objects.
- We initialize instance variables using constructors while creating an object. We can also use instance blocks to initialize the instance variables.

File Name: Person.java

```
public class Person {
 private String name;
 private int age;

 public Person(String name, int age) {
 this.name = name;
 this.age = age;
 }

 public void introduce() {
 System.out.println("My name is " + name + " and I am " + age + " years old.");
 }

 public static void main(String[] args) {
 Person person1 = new Person("John", 25);
 person1.introduce();

 Person person2 = new Person("Alice", 30);
 person2.introduce();
 }
}
```

Output:

```
My name is John and I am 25 years old.
My name is Alice and I am 30 years old.
```

In this example, we have a **Person** class with two instance variables: **name** and **age**. The **Person** class also has a constructor that takes **name** and **age** as parameters and assigns them directly to the instance variables using **this** keyword. The **introduce()** method is responsible for printing out a message that includes the **name** and **age** of the person.

In the **main()** method, we create two **Person** objects: **person1** and **person2**. We pass the **name** and **age** values directly to the constructor during object creation.

Finally, we call the **introduce()** method on each **Person** object, which prints out the message with their respective names and ages.

This example showcases the usage of instance variables to store data specific to each object of the class, and the constructor to initialize the instance variables during object creation. The **introduce()** method demonstrates the usage of the instance variables to display information about each person.

### 3. Static Variables

Static variables are also known as class variables.

- These variables are declared similarly to instance variables. The difference is that static variables are declared using the **static** keyword within a class outside of any method, constructor, or block.

- Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of a static variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for *Wrapper classes* like *Integer* it is *null*, etc.
- If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.
- If we access a static variable without the class name, the compiler will automatically append the class name. But for accessing the static variable of a different class, we must mention the class name as 2 different classes might have a static variable with the same name.
- Static variables cannot be declared locally inside an instance method.

```
import java.io.*;
class Emp {
 public static double salary;
 public static String name = "Tuhin";
}

public class EmpDemo {
 public static void main(String args[]){
 Emp.salary = 1000;
 System.out.println(Emp.name);
 System.out.println(Emp.salary);
 }
}
```

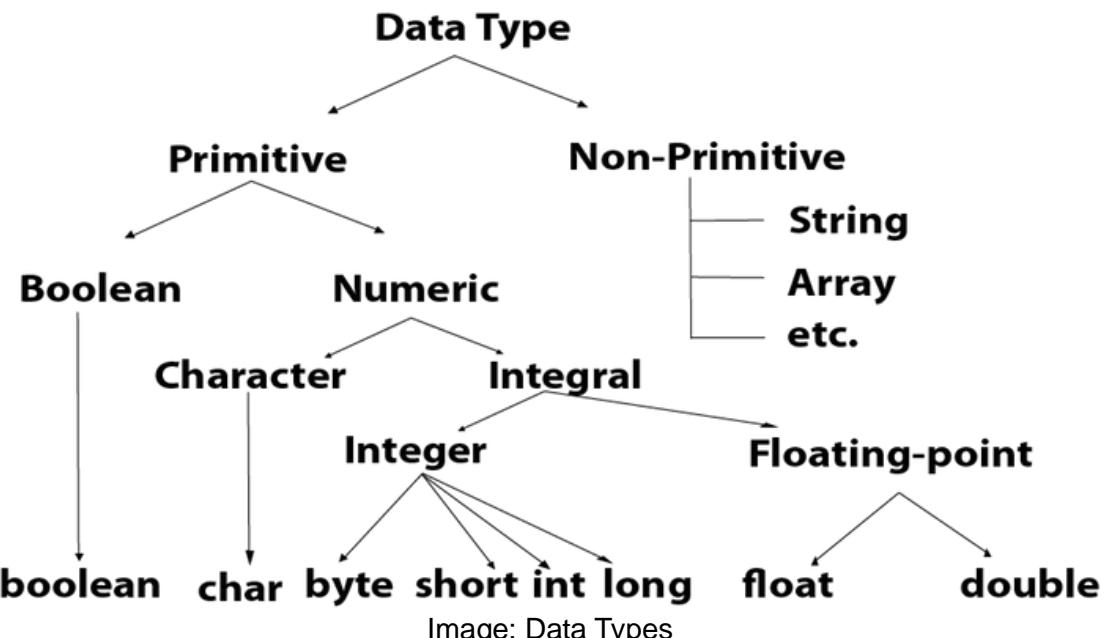
Output:

```
Hari
1000
```

## Data Type

Data types indicate the specific sizes and values that can be stored in the flexible. There are two variety of data variety in Java programming:

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



### What are Primitive Data Types In Java?

In Java language, untrained data variety are the building blocks of data consumption. These are the most vital data variety available in Java language. Java is a statically-typed programming language. It means, all flexible must be announced before its use. That is why we need to reveal flexible variety and name. There are 8 variety of untrained data variety:

- boolean
- byte
- char
- short
- int
- long
- float
- double

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

### What is Boolean Data Type?

The Boolean data variety is used to store only two probable values: true and false. This data variety is used for elementary flags that track true/false conditions. The

Boolean data variety indicate one bit of information, but its "size" can't be prominent precisely.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 boolean a = true;
 boolean b = false;
 System.out.println(a);
 System.out.println(b);
 }
}
```

Output:

```
true
false
```

## What is Byte Data Type?

The byte data variety is an example of untrained data type. It is an 8-bit signed two's accompaniment integer. Its amount- variety lies between -128 to 127 (comprehensive). Its minimum amount is -128 and maximum amount is 127.

The byte data variety is used to recover memory in big arrays where the memory savings is most required. It recovers capacity because a byte is 4 times smaller than an integral. It can also be preloved in place of "int" data variety.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 byte a = 2;
 byte b = -4;
 System.out.println(a);
 System.out.println(b);
 }
}
```

Output:

```
2
-4
```

## What is Short Data Type?

The short data variety is a 16-bit signed two's complement integer. Its amount-variety lies between -32,768 to 32,767 (comprehensive). Its minimum amount is -32,768 and maximum amount is 32,767. Its default amount is 0.

The short data variety can also be used to recover memory just like byte data variety. A, abrupt data variety is 2 times smaller than an integer.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 short a = 2000;
 short b = -2000;
 System.out.println(a);
 System.out.println(b);
 }
}
```

Output:

```
2000
-2000
```

## What is Int Data Type?

The int data type is a 32-bit signed two's complement integer. Its amount types lies between  $-2,147,483,648$  ( $-2^{31}$ ) to  $2,147,483,647$  ( $2^{31}-1$ ) (inclusive). Its minimum amount is  $-2,147,483,648$  and maximum amount is  $2,147,483,647$ . The int data types is generally preloved as a default data type for integral amount unless if there is no problem about memory.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 int a = 200000;
 int b = -300000;
 System.out.println(a);
 System.out.println(b);
 }
}
```

Output:

```
200000
-300000
```

## What is Long Data Type?

The long data types is a 64-bit two's complement integer. Its amount type lies between  $-9,223,372,036,854,775,808(-2^{63})$  to  $9,223,372,036,854,775,807(2^{63}-1)$ .

<sup>1)</sup>(comprehensive). Its minimum amount is - 9,223,372,036,854,775,808 and maximum amount is 9,223,372,036,854,775,807. Its default amount is 0. The long data variety is used when you need a range of values more than those distribute by int.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 long a = 200000L;
 long b = -300000L;
 System.out.println(a);
 System.out.println(b);
 }
}
```

**Output:**

```
200000L
-300000L
```

## What is Float Data?

The float data types is a single-precision 32-bit IEEE 754 floating point. Its amount range is unlimited. It is recommended to use a float (instead of double) if you need to recover memory in large arrays of floating point numbers. The float data types should never be preloved for precise values, such as currency. Its default amount is 0.0F.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 float a = -2.00;
 float b = 2.00;
 System.out.println(a);
 System.out.println(b);
 }
}
```

**Output:**

```
-2.00
2.00
```

## What is Double Data Type?

The double data types is a double-precision 64-bit IEEE 754 floating point. Its amount range is endless. The double data variety is generally preloved for decimal amount just like float. The double data type also should never be preloved for precise amount, such as currency. Its default amount is 0.0d.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 double a = -2.00;
 double b = 2.00;
 System.out.println(a);
 System.out.println(b);
 }
}
```

**Output:**

```
-2.00
2.00
```

## What is Char Data Type?

The char data type is a single 16-bit Unicode character. Its amount type lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data is type preloved to store characters.

**Example:**

```
import java.io.*;
class Simple{
 public static void main(String args[]){
 char a = 'D';
 char b = 'C';
 System.out.println(a);
 System.out.println(b);
 }
}
```

**Output:**

```
D C
```

## Java Keywords

A keyword in programming is a reserved word that has a special meaning and predefined functionality within the programming language. These words are part of the language's syntax and cannot be used as identifiers (variable names, method names, etc.) by the programmer.

Keywords are used to define the structure, behavior, and flow of a program. They provide specific instructions or declarations that the compiler or interpreter understands and executes accordingly. Keywords are often used to define control structures, data types, access levels, loops, conditional statements, and more.

The usage and functionality of keywords are determined by the programming language itself. Each programming language has its own set of keywords with specific meanings and rules. Attempting to use a keyword as an identifier or using a keyword in a way that contradicts its predefined usage will result in a syntax error.

Here is a list of Java keywords:

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert</code>	<code>default</code>	<code>if</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>goto</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp</code>	<code>volatile</code>
<code>const</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

These keywords have specific meanings and usages in the Java language. For example:

- `if, else, switch, case, default`: Used for conditional and control flow statements.
- `for, while, do`: Used for loop constructs.
- `class, interface, enum`: Used for defining classes, interfaces, and enumerations, respectively.
- `public, private, protected`: Used for defining access levels for classes, methods, and variables.
- `static, final, abstract`: Used for defining static members, constant values, and abstract methods or classes, respectively.
- `return`: Used for returning values from methods.
- `void`: Used as a return type for methods that do not return a value.

## Java Operators

Operator in Java is a symbol or a sign which is used for performing operations. For example: `-`, `+`, `/`, `*` etc. In Java there are many different types of operators which are given below: -

- Assignment Operator
- Arithmetic Operator
- Relational Operator
- Logical Operator
- Bitwise Operator
- Unary Operator
- Ternary Operator

### Assignment Operators

Assignment = operators are used to assigns the value on its left variable to the right-side variable or data. Assignment operators in Java are given below:

Operator	Example	Equal To	Explain
=	a = b;	a = b;	It takes the value of b and assigns into a.
+=	a += b;	a = a + b;	It takes the value of a + b and assigns into a.
-=	a -= b;	a = a - b;	It takes the value of a - b and assigns into a.
*=	a *= b;	a = a * b;	It takes the value of a * b and assigns into a.
/=	a /= b;	a = a / b;	It takes the value of a / b and assigns into a.
%=	a %= b;	a = a % b;	It takes the value of a % b and assigns into a.

### Example:

```
import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 12, b = 5;
 int c = 12, d = 5;

 a += b;
 c %= d;

 System.out.println(a);
 System.out.println(c);
 }
}
```

Output :

```
17
2
```

### Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on two or more variables. Arithmetic operators in Java are given below:

Operator	Operation
+	It adds two or more variables / values one with others.
-	It subtracts two or more variables / values on from another
*	It multiplies two or more variables / values one with others.
/	It division two or more variables / values one by another. [NB: If you use the division operator with two integers, then the result will also be an integer. And, if atleast one of the operands is a floating point number, you will get the result will also be in floating point.]
%	Modulo Operation is basically keep the remainder.

### Example:

```
import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 12, b = 5;
 float c = 12, d = 5;
 int Addition = a + b; // addition operator
 int Subtract = a - b; // subtraction operator
 int Multiplication = a * b; // multiplication operator
 float Division = c / d; // division operator
 }
}
```

```

int Modulo = a % b; // modulo operator

System.out.println("a + b = "+ Addition);
System.out.println("a - b = "+ Substract);
System.out.println("a * b = "+ Multiplication);
System.out.println("c / d = "+ Division);
System.out.println("a % b = "+ Modulo);
}
}

```

Output :

```

a + b = 17
a - b = 5
a * b = 60
c / d = 2.4
a % b = 2

```

### ***Relational Operators***

Relational operators are used to check the relationship or comparison between two operands or variables. Relational operators in Java are given below:

Operator	Description	Example
<b>==</b>	<b>Is Equal To</b>	5 == 5 will return true.
<b>!=</b>	<b>Is Not Equal To</b>	5 != 4 will return true.
<b>&gt;</b>	<b>Greater Than</b>	5 > 4 will return true.
<b>&lt;</b>	<b>Less Than</b>	5 < 4 will return false.
<b>&gt;=</b>	<b>Greater Than Equal To</b>	5 >= 4 will return true.
<b>&lt;=</b>	<b>Less Than Equal To</b>	5 <= 4 will return false.

### ***Example:***

```

import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 5, b = 4;
 System.out.println(a == b);
 System.out.println(a >= b);
 }
}

```

Output :

```

false
true

```

### ***Logical Operators***

Logical operators are used to check 2 or more expression is true or not. Logical operators in Java are given below:

Operator	Example	Description
<b>&amp;&amp; (Logical AND)</b>	5 > 4 && 9 > 3	If both side expression of && (AND) is true, then it returns true.
<b>   (Logical OR)</b>	5 > 4    9 < 3	If any one side expression of    (OR) is true, then it returns true.

<b>! (Logical NOT)</b>	If expression is true, then it returns false and if expression is false then it returns true and vice versa.
------------------------	--------------------------------------------------------------------------------------------------------------

### Example:

```
import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 5, b = 4, c = 1;
 System.out.println(a > b && b > c);
 System.out.println(a > b || b < c);
 System.out.println(!(a > b));
 }
}
```

Output :

```
true
true
false
```

### Bitwise Operators

Bitwise operators in Java are used to perform operations on every bits. Bitwise operators in java are:

Operator	Description
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise Complement
<<	Bitwise Left Shift
>>	Bitwise Right Shift
>>>	Unsigned Right Shift

### Unary Operators

Unary operators are worked with one variable or operands. Unary operators in Java are given below:

Operator	Description	Example
++	Increment Operator	If a = 4 then a++ will be 5.
--	Decrement Operator	If a = 4 then a-- will be 3.

### Example:

```
import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 5, b = 4;
 System.out.println(a++);
 System.out.println(b--);
 }
}
```

Output :

### Ternary Operators

Ternary operators is basically a conditional operator and sign is "?". For an example:

**variable = expression? value1: value2;**

In the above example if expression is true then value1 will be stored in variable and if expression is false then value2 will be stored in variable.

#### Example:

```
import java.io.*;
Class Main{
 Public static void main(String args[]){
 int a = 5, b;
 b = a < 3 ? 2 : 4
 System.out.println(b);
 }
}
```

Output :

4

## Input and Output

### Java Output

In Java, you can use the **System.out** object to perform output operations. The **System.out** object is an instance of the **PrintStream** class and provides methods for printing output to the console. Here are some commonly used methods for output in Java:

#### 1. **print()**:

- This method prints the given data to the console without appending a newline character.
- Example: **System.out.print("Hello")**;

#### 2. **println()**:

- This method prints the given data to the console and appends a newline character at the end.
- Example: **System.out.println("Hello")**;

#### 3. **printf()**:

- This method allows you to format the output using format specifiers.
- Example: **System.out.printf("Name: %s, Age: %d", name, age)**;

Here's a complete example that demonstrates these output methods:

#### Example:

```
public class Main {
 public static void main(String[] args) {
 String name = "John";
 int age = 25;
```

```
 System.out.print("Hello");
 System.out.println(" World!");
 System.out.printf("Name: %s, Age: %d", name, age);
 }
}
```

Output :

```
Hello World!
Name: John, Age: 25
```

In the example above, we use the **print()** method to print "Hello" without a newline character, the **println()** method to print "World!" with a newline character, and the **printf()** method to format and print the name and age with specific placeholders **%s** and **%d**, respectively.

You can use these output methods to display results, messages, and other information to the console during the execution of your Java program.

### Difference between **println()**, **print()** and **printf()**

- **print()** - It prints string inside the quotes.
- **println()** - It prints string inside the quotes similar like **print()** method. Then the cursor moves to the beginning of the next line.
- **printf()** - It provides string formatting (similar to **printf** in C/C++ programming).

## Java Input

Java provides different ways to get input from the user. However, in this tutorial, you will learn to get input from user using the object of Scanner class.

In order to use the object of Scanner, we need to import `java.util.Scanner` package.

```
import java.util.Scanner;
```

To learn more about importing packages in Java, visit [Java Import Packages](#). Then, we need to create an object of the Scanner class. We can use the object to take input from the user.

```
// create an object of Scanner
Scanner input = new Scanner(System.in);
// take input from the user
int number = input.nextInt();
```

### Example: Get Integer Input From the User

```
import java.util.Scanner;
class Input {
 public static void main(String[] args) {
 Scanner input = new Scanner(System.in);
 System.out.print("Enter an integer: ");
 int number = input.nextInt();
 System.out.println("You entered " + number);
```

```
 // closing the scanner object
 input.close();
}
}
```

Output:

```
Enter an integer: 23 You entered 23
```

In the above example, we have created an object named `input` of the `Scanner` class. We then call the `nextInt()` method of the `Scanner` class to get an integer input from the user. Similarly, we can use `nextLong()`, `nextFloat()`, `nextDouble()`, and `next()` methods to get long, float, double, and string input respectively from the user.

**Note:** We have used the `close ()` method to close the object. It is recommended to close the scanner object once the input is taken.

### Example: Get float, double and String Input

```
import java.util.Scanner;
class Input {
 public static void main(String[] args) {
 Scanner input = new Scanner(System.in);
 // Getting float input
 System.out.print("Enter float: ");
 float myFloat = input.nextFloat();
 System.out.println("Float entered = " + myFloat);
 // Getting double input
 System.out.print("Enter double: ");
 double myDouble = input.nextDouble();
 System.out.println("Double entered = " + myDouble);
 // Getting String input
 System.out.print("Enter text: ");
 String myString = input.next();
 System.out.println("Text entered = " + myString); }
}
```

Output:

```
Enter float: 2.343
Float entered = 2.343
Enter double: -23.4
Double entered = -23.4
Enter text: Hey!
Text entered = Hey!
```

## Java Expressions & Blocks

### Java Expression

An expression is a construct which is made up of literals, variables, method calls and operators following the syntax of Java. Every expression consists of at least one operator and an operand. Operand can be either a literal, variable or a method invocation.

Following are some of the examples for expressions in Java:

```
int a = 10; //Assignment expression

System.out.println("Value = "+x);

int result = a + 10; //Assignment exp

if(val1 <= val2) //Boolean expression

b = a++; //Assignment exp
```

## How expressions are evaluated?

It is common for an expression to have more than one operator. For example, consider the below example:

**(20 \* 5) + (10 / 2) – (3 \* 10)**

So, how is the above expression evaluated? Expression evaluation in Java is based upon the following concepts:

- Type promotion rules
- Operator precedence
- Associativity rules

## Operator precedence

All the operators in Java are divided into several groups and are assigned a precedence level. The operator precedence chart for the operators in Java is shown below:

Highest Precedence	Operators
	<code>++ (postfix), -- (postfix)</code>
	<code>++ (prefix), -- (prefix), ~, !, +(unary), -(unary), (type-cast)</code>
	<code>*, /, %</code>
	<code>+, -</code>
	<code>&gt;&gt;, &gt;&gt;&gt;, &lt;&lt;</code>
	<code>&gt;, &gt;=, &lt;, &lt;=, instanceof</code>
	<code>==, !=</code>
	<code>&amp;</code>
	<code>^</code>
	<code> </code>
	<code>&amp;&amp;</code>
	<code>  </code>
	<code>?:</code>
Lowest Precedence	<code>=, op=</code>

Image: Data Types  
Reference: <https://www.javatpoint.com/java-data-types>

Now let's consider the following expression:

## **10 – 2 \* 5**

One will evaluate the above expression normally as, 10-2 gives 8 and then 8\*5 gives 40. But Java evaluates the above expression differently. Based on the operator precedence chart shown above, \* has higher precedence than +. So,  $2^* 5$  is evaluated first which gives 10 and then  $10 - 10$  is evaluated which gives 0.

## **Associativity Rules**

When an expression contains operators from the same group, associativity rules are applied to determine which operation should be performed first. The associativity rules of Java are shown below:

Operator Group	Associativity	Type of Operation
<code>! ~ ++ -- + -</code>	right-to-left	unary
<code>* / %</code>	left-to-right	multiplicative
<code>+ -</code>	left-to-right	additive
<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>	left-to-right	bitwise
<code>&lt; &lt;= &gt; &gt;=</code>	left-to-right	relational
<code>== !=</code>	left-to-right	relational
<code>&amp;</code>	left-to-right	bitwise
<code>^</code>	left-to-right	bitwise
<code> </code>	left-to-right	bitwise
<code>&amp;&amp;</code>	left-to-right	boolean
<code>  </code>	left-to-right	boolean
<code>?:</code>	right-to-left	conditional
<code>= += -= *= /= %= &amp;=</code>	right-to-left	assignment
<code>^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>	right-to-left	assignment
<code>,</code>	left-to-right	comma

Image: Data Types

Reference: <https://www.javatpoint.com/java-data-types>

Now, let's consider the following expression:

### **10-6+2**

In the above expression, the operators + and – both belong to the same group in the operator precedence chart. So, we have to check the associativity rules for evaluating the above expression. Associativity rule for + and – group is left-to-right i.e., evaluate the expression from left to right. So, 10-6 is evaluated to 4 and then 4+2 is evaluated to 6.

## **Use of parenthesis in expressions**

Let's look at our original expression example:

### **(20 \* 5) + (10 / 2) – (3 \* 10)**

You might think that, what is the need of parenthesis (and) in the above expression. The reason I had included them is, parenthesis have the highest priority (precedence) over all the operators in Java.

So, in the above expression,  $(20*5)$  is evaluated to 100,  $(10/2)$  is evaluated to 5 and  $(3*10)$  is evaluated to 30. Now, our intermediate expression looks like:

### **100 + 5 – 30**

Now, we can apply the associativity rules and evaluate the expression. The final answer for the above expression is 75. There is another popular use of parenthesis.

We will use them in print statements. For example, consider the following piece of code:

```
int a=10, b=20;
System.out.println("Sum of a and b is: "+a+b);
```

One might think that the above code will produce the output: "Sum of a and b is: 30". The real output will be:

Sum of a and b is: 1020

```
int a=10, b=20;
System.out.println("Sum of a and b is: "+(a+b));
```

Now  $(a+b)$  is evaluated first and then concatenated to the rest.

## Java Blocks

A block is a group of statements (zero or more) that is enclosed in curly braces { }.

For example,

```
class Main {
 public static void main(String[] args) {

 String band = "Beatles";

 if (band == "Beatles") { // start of block
 System.out.print("Hey ");
 System.out.print("Jude!");
 } // end of block
 }
}
```

### Output:

Hey Jude!

In the above example, we have a block if {....}.

Here, inside the block we have two statements:

```
System.out.print("Hey ");
System.out.print("Jude!");
```

However, a block may not have any statements. Consider the following examples,

```
class Main {
 public static void main(String[] args) {
 if (10 > 5) { // start of block
 } // end of block
 }
}
```

This is a valid Java program. Here, we have a block if {...}. However, there is no any statement inside this block.

```
class AssignmentOperator {
 public static void main(String[] args) { // start of block } // end of block
}
```

Here, we have block public static void main() {...}. However, similar to the above example, this block does not have any statement.

### Comment

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

## Single-line Comments

Single-line comments start with two forward slashes (//). Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code: Example

```
// This is a comment
System.out.println("Hello World");
System.out.println("Hello World"); // This is a comment
```

## Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/. Any text between /\* and \*/ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

```
public class Main {
 public static void main(String[] args) {
 /* The code below will print the words Hello World
 * to the screen, and it is amazing */
 System.out.println("Hello World");
 }
}
```

## Java Control Statements

In Java, control statements are used to control the flow of execution in a program. They allow you to make decisions, repeat a block of code multiple times, and jump to different sections of code based on certain conditions. Here are the commonly used control statements in Java:

### if statement:

The if statement is used to execute a block of code only if a certain condition is true. It has the following syntax:

```
if (condition) {
 // code to be executed if the condition is true
}
```

Example:

```
int age = 25;

if (age >= 18) {
 System.out.println("You are eligible to vote.");
}
```

Output:

You are eligible to vote.

Explanation:

In the example above, we have a variable called `age` initialized with the value 25. The if statement checks whether the age is greater than or equal to 18, indicating eligibility to vote.

Here's how the if statement works:

1. The condition `age >= 18` checks if the value of `age` is greater than or equal to 18. If the condition evaluates to true, it means the person is 18 years or older and eligible to vote.
2. Since the value of `age` is 25 (which is greater than 18), the condition is true, and the code block within the if statement is executed.
3. The code block within the if statement prints the message "You are eligible to vote." using the **System.out.println()** statement.
4. As a result, the output of the program is "You are eligible to vote."

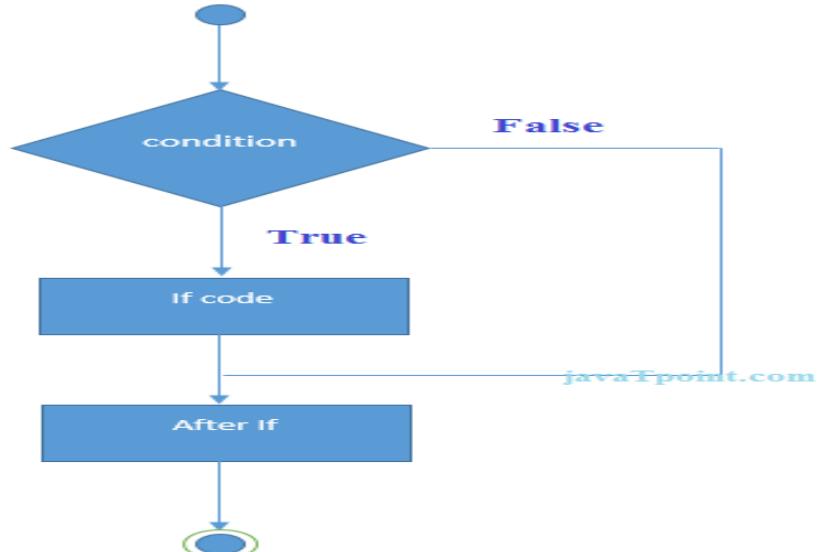
This example demonstrates how the if statement allows you to execute a block of code only if a certain condition is true. If the condition specified in the if statement evaluates to true, the code within the if block is executed. Otherwise, if the condition is false, the code block is skipped entirely. This allows you to selectively execute code based on specific conditions in your program.

### **if-else statement:**

The if-else statement allows you to execute one block of code if a condition is true, and another block of code if the condition is false. It has the following syntax:

```
if (condition) {
 // code to be executed if the condition is true
} else {
 // code to be executed if the condition is false
}
```

Flowchart:



Reference: [Java If else - Javatpoint](#)

**Example:**

```

int number = 10;

if (number % 2 == 0) {
 System.out.println("The number is even.");
} else {
 System.out.println("The number is odd.");
}

```

**Output:**

```
The number is even.
```

**Explanation:**

In the example above, we have a variable called **number** initialized with the value 10. The if-else statement checks whether the number is even or odd.

Here's how the if-else statement works:

1. The condition **number % 2 == 0** checks if the remainder of dividing **number** by 2 is equal to 0. If the condition evaluates to true, it means the number is divisible by 2 and hence even.
2. Since 10 is divisible by 2 (the remainder is 0), the condition is true, and the code block within the if statement is executed.
3. The code block within the if statement prints the message "The number is even." using the **System.out.println()** statement.
4. Since the if condition is true, the else block is skipped entirely.
5. As a result, the output of the program is "The number is even."

This example demonstrates how the if-else statement allows you to execute different blocks of code based on a condition. If the condition specified in the if statement is true, the code within the if block is executed. Otherwise, if the condition is false, the code within the else block is executed. This enables you to perform different actions based on different conditions in your program.

### **if-else if-else statement:**

The if-else if-else statement allows you to test multiple conditions and execute different blocks of code based on the results. It has the following syntax:

```
if (condition1) {
 // code to be executed if condition1 is true
} else if (condition2) {
 // code to be executed if condition2 is true
} else {
 // code to be executed if all conditions are false
}
```

Example:

```
int num = 0;
if (num > 0) {
 System.out.println("The number is positive.");
} else if (num < 0) {
 System.out.println("The number is negative.");
} else {
 System.out.println("The number is zero.");
}
```

Output:

```
The number is zero.
```

Explanation:

In the example above, we have a variable called `num` initialized with the value 0. The if-else if-else statement checks whether the number is positive, negative, or zero.

Here's how the if-else if-else statement works:

1. The first condition `num > 0` checks if the value of `num` is greater than 0. If the condition evaluates to true, it means the number is positive.
2. Since the value of `num` is 0, the first condition is false. The program moves to the next condition.
3. The second condition `num < 0` checks if the value of `num` is less than 0. If the condition evaluates to true, it means the number is negative.
4. Since the value of `num` is 0, the second condition is also false. The program moves to the else block.
5. The else block is executed when none of the previous conditions are true. In this case, since the value of `num` is 0, the else block is executed.
6. The code block within the else statement prints the message "The number is zero." using the `System.out.println()` statement.
7. As a result, the output of the program is "The number is zero."

This example demonstrates how the if-else if-else statement allows you to test multiple conditions and execute different blocks of code based on the results. It provides a way to handle various scenarios and choose different actions based on different conditions in your program.

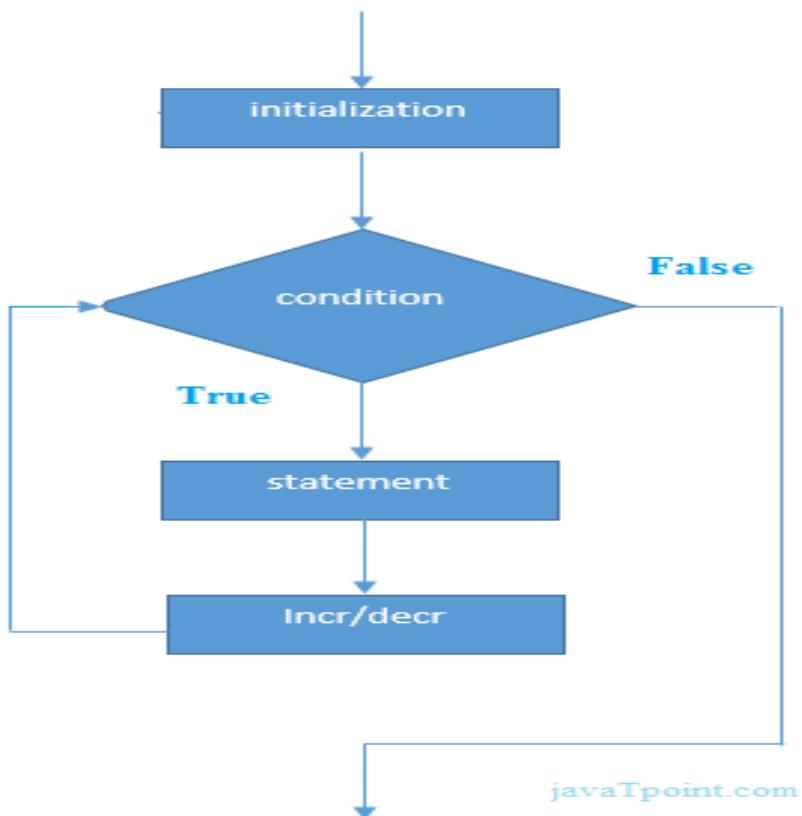
## For loop

The "for" loop is a control statement in Java that allows you to repeatedly execute a block of code for a specified number of times. It provides a concise way to initialize,

test a condition, and update a variable in a single line. Here's the general syntax of a "for" loop:

```
for (initialization; condition; update) {
 // code to be executed
}
```

Flowchart:



Reference: [Loops in Java | Java For Loop \(Syntax, Program, Example\) - Javatpoint](#)

Let's break down the components of the "for" loop:

**Initialization:** It is an optional statement that is executed once before the loop starts. It is typically used to initialize a loop control variable. For example, `int i = 0` initializes the variable `i` to 0.

**Condition:** It is an expression that is evaluated before each iteration of the loop. If the condition is true, the loop continues to execute. If the condition is false, the loop terminates. For example, `i < 10` checks whether `i` is less than 10.

**Update:** It is an optional statement that is executed after each iteration of the loop. It is commonly used to update the loop control variable. For example, `i++` increments the value of `i` by 1 after each iteration.

**Code to be executed:** It is the block of code that is executed repeatedly as long as the condition is true. This block of code is enclosed within curly braces `{}`.

Example:

Prints the numbers 1 to 5:

```
for (int i = 1; i <= 5; i++) {
 System.out.println(i);
}
```

Output:

```
1
2
3
4
5
```

In the example above, the loop initializes `i` to 1, checks if `i` is less than or equal to 5, executes the code within the loop (printing the value of `i`), and updates `i` by incrementing it by 1 after each iteration.

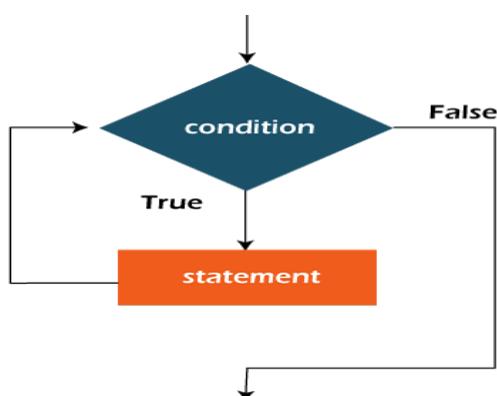
The "for" loop is useful when you know the number of iterations in advance and want to iterate over a sequence of values or perform a specific task repeatedly.

### **while loop:**

The while loop executes a block of code repeatedly as long as a given condition is true. It has the following syntax:

```
while (condition) {
 // code to be executed
}
```

Flowchart:



Reference: [Java while loop - Javatpoint](#)

Example:

```
int count = 1;

while (count <= 5) {
 System.out.println("Count: " + count);
 count++;
}
```

Output:

```
Count: 1
Count: 2
```

```
Count: 3
Count: 4
Count: 5
```

#### Explanation:

In the example above, we have a variable called `count` initialized with the value 1. The while loop is used to repeatedly execute a block of code as long as the condition `count <= 5` is true.

Here's how the while loop works:

1. The condition `count <= 5` checks if the value of `count` is less than or equal to 5. If the condition evaluates to true, the code block within the while loop is executed.
2. Initially, the value of `count` is 1, which is less than or equal to 5. Therefore, the condition is true, and the code block within the while loop is executed.
3. The code block within the while loop prints the value of `count` along with the string "Count: " using the `System.out.println()` statement.
4. After executing the code block, the value of `count` is incremented by 1 using the `count++` statement.
5. The program then goes back to the beginning of the while loop and checks the condition again.
6. This process continues until the condition `count <= 5` becomes false. Once the value of `count` reaches 6, the condition is false, and the loop terminates.
7. As a result, the output of the program is "Count: 1", "Count: 2", "Count: 3", "Count: 4", and "Count: 5", each on a separate line.

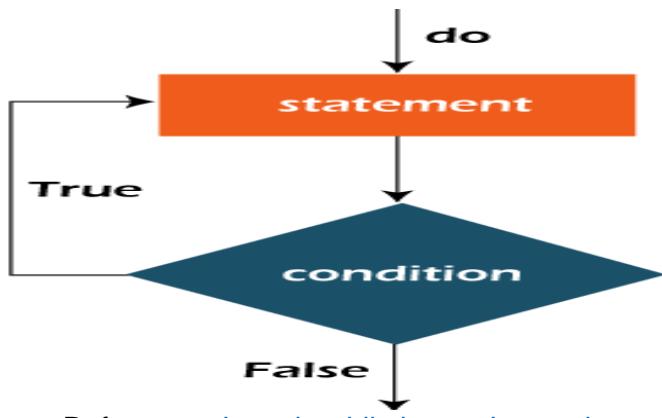
This example demonstrates how the while loop repeatedly executes a block of code as long as a specified condition is true. It is useful when you don't know the exact number of iterations in advance and want to keep executing a certain task until a certain condition is met.

#### do-while loop:

The do-while loop is similar to the while loop, but it ensures that the code block is executed at least once before checking the condition. It has the following syntax:

```
do {
 // code to be executed
} while (condition);
```

Flowchart:



Reference: [Java do while loop - Javatpoint](#)

**Example:**

```
int count = 1;
do {
 System.out.println("Count: " + count);
 count++;
} while (count <= 5);
```

**Output:**

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

**Explanation:**

In the example above, we have a variable called `count` initialized with the value 1. The do-while loop is used to repeatedly execute a block of code as long as the condition `count <= 5` is true. The key difference between a do-while loop and a while loop is that the do-while loop always executes the code block at least once, even if the condition is initially false.

Here's how the do-while loop works:

1. The code block within the do-while loop is executed first, without checking the condition.
2. The code block prints the value of `count` along with the string "Count: " using the `System.out.println()` statement.
3. After executing the code block, the value of `count` is incremented by 1 using the `count++` statement.
4. The program then checks the condition `count <= 5`. If the condition evaluates to true, the loop continues to execute. If the condition is false, the loop terminates.
5. In this case, the value of `count` is incremented to 2, which is less than or equal to 5. Therefore, the condition is true, and the loop continues to execute.
6. The process repeats until the condition `count <= 5` becomes false. Once the value of `count` reaches 6, the condition is false, and the loop terminates.
7. As a result, the output of the program is "Count: 1", "Count: 2", "Count: 3", "Count: 4", and "Count: 5", each on a separate line.

This example demonstrates how the do-while loop first executes the code block and then checks the condition. It ensures that the code block is executed at least once, regardless of the initial condition. The loop continues to execute as long as the specified condition remains true.

### **Switch statement:**

The switch statement is used to select one of many code blocks to be executed. It evaluates an expression and compares its value against various cases. If a match is found, the corresponding block of code is executed. It has the following syntax:

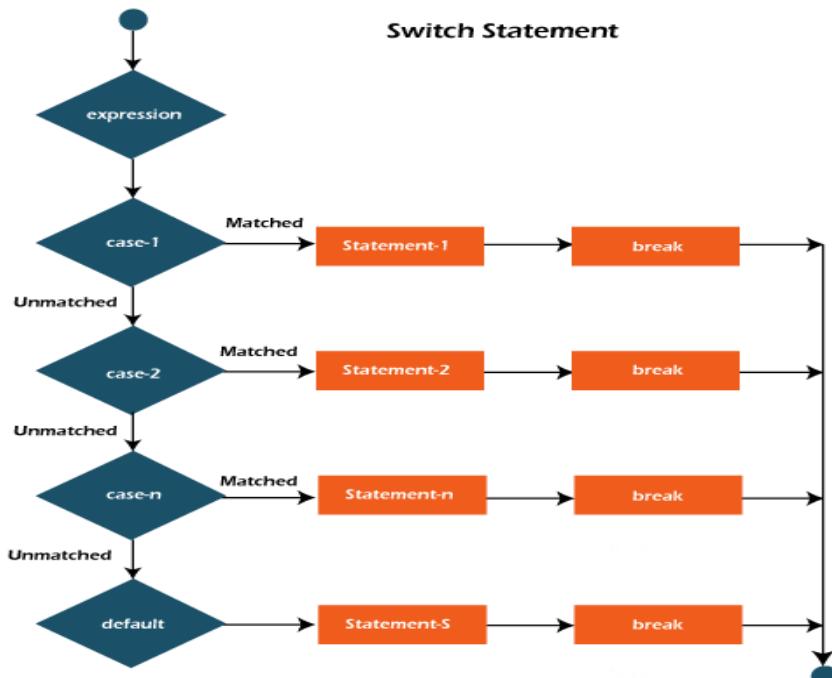
```
switch (expression) {
 case value1:
 // code to be executed if the expression matches value1
 break;
```

```

case value2:
 // code to be executed if the expression matches value2
 break;
// more cases...
default:
 // code to be executed if none of the cases match
 break;
}

```

Flowchart:



Reference: [Java Switch - Javatpoint](#)

Example:

```

public class SwitchExample {
 public static void main(String[] args) {
 //Declaring a variable for switch expression
 int number=20;
 //Switch expression
 switch(number){
 //Case statements
 case 10: System.out.println("10");
 break;
 case 20: System.out.println("20");
 break;
 case 30: System.out.println("30");
 break;
 //Default case statement
 default: System.out.println("Not in 10, 20 or 30");
 }
 }
}

```

```
}
```

Output:

```
20
```

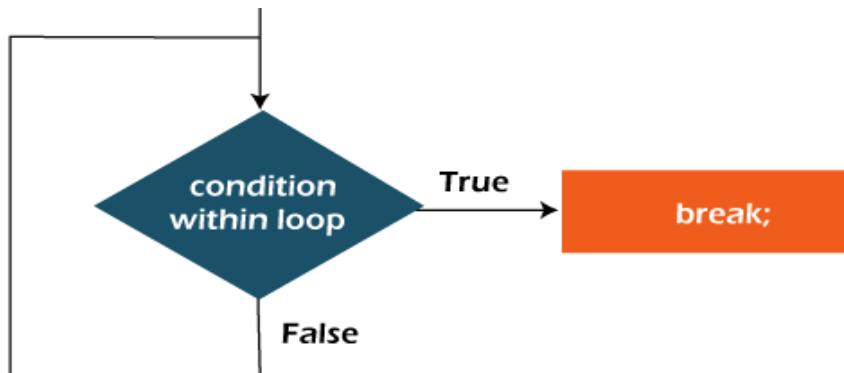
### break and continue statements:

In Java, the "break" and "continue" statements are used to alter the flow of control within loops (such as for loops, while loops, and do-while loops) and switch statements. Here's an explanation of each:

#### 1. break statement:

The "break" statement is used to immediately exit from the innermost loop or switch statement it is located in. When encountered, the program will continue executing from the next statement after the loop or switch. It is typically used to terminate a loop prematurely based on a certain condition.

Flowchart:



Reference: [Java Break – Javatpoint](#)

Example:

```
for (int i = 0; i < 10; i++) {
 if (i == 5) {
 break; // exit the loop when i equals 5
 }
 System.out.println(i);
}
```

Output:

```
0
1
2
3
4
```

In the example above, the loop terminates when the value of `i` becomes 5 because the "break" statement is encountered. As a result, only the numbers 0 to 4 are printed.

#### 2. continue statement:

The "continue" statement is used to skip the rest of the current iteration of a loop and move on to the next iteration. When encountered, the program jumps to the loop's

increment or update statement (for loops) or the loop's condition check (while loops and do-while loops). It is often used to skip certain iterations based on a specific condition.

Example:

```
for (int i = 0; i < 10; i++) {
 if (i % 2 == 0) {
 continue; // skip even numbers
 }
 System.out.println(i);
}
```

Output:

```
1
3
5
7
9
```

In the example above, the "continue" statement is used to skip the even numbers. When `i` is even (divisible by 2), the rest of the code in the loop is skipped, and the program moves on to the next iteration.

Both "break" and "continue" statements provide control over the execution flow within loops and switch statements, allowing you to customize the behaviour of your code based on specific conditions.

## Java Array & String:

### Java Arrays

Array is a collection of elements of same type. For example, an int array contains integer elements and a String array contains String elements. The elements of Array are stored in contiguous locations in the memory. This is how an array looks like:

```
int number [] = new int [10];
```

Here number is the array name. The type of the array is integer, which means it can store integer values. The size of the array is 10.

Array works on an index-based system. In the above array, number [0] represents the first element of the array, number [1] represents the second element of the array and so on. The index of array starts from 0 and ends at array\_size-1. In the above example, the index of first element is 0 and index of 10th element is 9.

### Advantages of Array

- **Better performance:** Since array works on a index based system, it is easier to search an element in the array, thus it gives better performance for various operations.
- **Multidimensional:** Unlike ArrayList which is single dimensional, arrays are multidimensional such as 2D array, 3D array etc.
- **Faster access:** Accessing an element is easy in array.

### Disadvantages of Array:

- **Fixed Size:** The size of the array is fixed, which cannot be increased later.

- **Allows only similar type elements:** Arrays are homogeneous, they don't allow different type values, for example an int array cannot hold string elements, similarly a String array cannot hold integer elements. Array is a collection of elements of same type.

For example, an int array contains integer elements and a String array contains String elements. The elements of Array are stored in contiguous locations in the memory.

### **Insertion and delegation require shifting of elements.**

### **Declaration, Instantiation and Initialization of Array in Java**

This is how we declare, instantiate and initialize an array.

```
int number[]; //array declaration
number[] = new int[10]; //array instantiation
number[0] = 10; //array Initialization
number[1] = 20; //array Initialization
```

We can also declare an array like this: All the three following syntax are valid for array declaration.

```
int[] number;
int []number;
int number[];
```

### **Example:**

The following example demonstrates, how we declared an int array, initialized it with integers and print the elements of the array using for loop.

Note: You can see that we have used length property of array to find the size of the array. The length property of array returns the number of elements present in the array.

### **Example:**

```
public class JavaExample{
 public static void main(String args[]){
 //array declaration, instantiation and initialization
 int number[] = {11, 22, 33, 44, 55};

 //print array elements
 //length property return the size of the array
 for(int i=0;i<number.length;i++)
 System.out.println("number["+i+"]: "+number[i]);
 }
}
```

### **Output:**

```
number[0]:11
number[1]:22
number[2]:33
number[3]:44
number[4]:55
```

### **Types of array in Java**

1. Single Dimensional Array

## 2. Multidimensional Array

### 1. Single dimensional array

#### Example:

```
public class JavaExample{
 public static void main(String args[]){
 //array declaration
 String names[] = new String[3];

 //array initialization
 names[0]="Rani";
 names[1]="Raja";
 names[2]="Reeta";
 //print array elements
 for(int i=0;i<names.length;i++)
 System.out.println("names["+i+"]: "+names[i]);
 }
}
```

#### Output:

```
names[0]: Chaitanya
names[1]: Ajeet
names[2]: Rahul
```

## 2. Multidimensional array

### Multidimensional array declaration:

This is how you can declare a multidimensional array: All the four syntax are valid multidimensional array declaration.

```
int[][] arr;
int [][]arr;
int arr[][];
int []arr[];
```

### Instantiate Multidimensional Array in Java

Number of elements in multidimensional array = number of rows\*number of columns.

The following array can store upto  $2 \times 3 = 6$  elements.

```
int[][] arr=new int[2][3]; //2 rows and 3 columns
```

### Initialize Multidimensional Array in Java

```
arr[0][0]=11;
arr[0][1]=22;
arr[0][2]=33;
arr[1][0]=44;
arr[1][1]=55;
arr[1][2]=66;
```

#### Example:

```

public class JavaExample{
 public static void main(String args[]){
 //two rows and three columns
 int arr[][]={{11,22,33},{44,55,66};

 //outer loop 0 till number of rows
 for(int i=0;i<2;i++){
 //inner loop from 0 till number of columns
 for(int j=0;j<3;j++){
 System.out.print(arr[i][j]+" ");
 }
 //new line after each row
 System.out.println();
 }
 }
}

```

**Output:**

```

11 22 33
44 55 66

```

**Print an Array elements using for-each loop**

There is another way to print Array elements without using array length property.

```

public class JavaExample{
 public static void main(String args[]){
 //String array
 String names[]{"Chaitanya", "Ajeet", "Rahul", "Hari"};

 //print array elements using for-each loop
 for(String str:names)
 System.out.println(str);

 //int array
 int numbers[]={1, 2, 3, 4, 5};

 //print array elements using for-each loop
 for(int num:numbers)
 System.out.println(num);
 }
}

```

**Output:**

```

Chaitanya
Ajeet
Rahul
Hari
1

```

2  
3  
4  
5

### Exception: **ArrayIndexOutOfBoundsException**

**ArrayIndexOutOfBoundsException** occurs when we access an array with an invalid index. This happens when the index is either negative or greater than or equal to the size of the array.

```
public class JavaExample{
 public static void main(String args[]){
 int number[]={1, 5, 7, 9, 11};
 for(int i=0;i<=number.length;i++){
 System.out.println(number[i]);
 }
 }
}
```

### Output:

ArrayIndexOutOfBoundsException in Java

## Java String

In Java, the **String** class represents a sequence of characters. It is one of the most commonly used classes in Java and is part of the **java.lang** package, which is automatically imported into every Java program. Here are some key points about **String** in Java:

1. **Immutable:** Strings in Java are immutable, meaning their values cannot be changed once they are created. Any operation that appears to modify a **String** actually creates a new **String** object with the modified value.
2. **String Literal:** Strings can be created using string literals enclosed in double quotes, such as "**Hello, World!**". String literals are stored in a string pool, and if multiple strings with the same value are created, they will reference the same object in the pool for memory optimization.
3. **String Concatenation:** Strings can be concatenated using the **+** operator or the **concat()** method. The **+** operator is commonly used for concatenation, where it combines multiple strings into a single string.
4. **String Methods:** The **String** class provides numerous methods to perform operations on strings, such as extracting substrings, searching for characters or substrings, replacing characters, converting case, and more. Some commonly used methods include **length()**, **charAt()**, **substring()**, **indexOf()**, **replace()**, **toLowerCase()**, **toUpperCase()**, etc.
5. **Equality Comparison:** To compare the content of two **String** objects, you should use the **equals()** method rather than the **==** operator. The **equals()** method compares the actual content of the strings, while the **==** operator checks if the two objects reference the same memory location.
6. **String Conversion:** Java provides the **toString()** method to convert objects to their string representation. Additionally, the **valueOf()** static methods in the

**String** class allow conversion from primitive types, arrays, and other objects to strings.

Here's a simple example demonstrating some string operations in Java:

```
public class Main {
 public static void main(String[] args) {
 String greeting = "Hello";
 String name = "John";

 // Concatenation
 String message = greeting + ", " + name + "!";
 System.out.println(message);

 // Length
 int length = message.length();
 System.out.println("Length: " + length);

 // Substring
 String substring = message.substring(7);
 System.out.println("Substring: " + substring);

 // Equality comparison
 boolean isEqual = greeting.equals(name);
 System.out.println("Equality: " + isEqual);
 }
}
```

#### Output

```
Hello, John!
Length: 13
Substring: John!
Equality: false
```

In the example above, we create **String** objects, perform concatenation using the **+** operator, get the length of a string, extract a substring, and compare string equality using the **equals()** method. These are just a few of the many operations you can perform with strings in Java.

## 2.3 Java OOPs Concepts

### Java Class and Objects

#### Object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

#### An object has three characteristics:

- State: represents the data (value) of an object.
- Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

### Objects: Real World Examples



Image: Object

Reference: <https://www.javatpoint.com/object-and-class-in-java>

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior. An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

#### Object Definitions:

- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behavior.
- The object is an instance of a class.

#### Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

#### Examples of states and behaviors

##### Example 1:

**Object:** House

**State:** Address, Color, Area

**Behavior:** Open door, close door

So, if I had to write a class based on states and behaviours of House. I can do it like this: States can be represented as instance variables and behaviours as methods of the class. We will see how to create classes in the next section of this guide.

```
class House {
 String address;
 String color;
```

```

double are;
void openDoor() {
 //Write code here
}
void closeDoor() {
 //Write code here
}
...
...
}

```

**Example 2:**

**Object:** Car

**State:** Color, Brand, Weight, Model

**Behavior:** Break, Accelerate, Slow Down, Gear change.

**Syntax:**

```

class <class_name>{
 field;
 method;
}

```

Here, fields (variables) and methods represent the state and behavior of the object respectively.

- fields are used to store data
- methods are used to perform some operations

For our bicycle object, we can create the class as

```

class Bicycle {
// state or field
private int gear = 5;

// behavior or method
public void braking() {
 System.out.println("Working of Braking");
}
}

```

In the above example, we have created a class named Bicycle. It contains a field named gear and a method named braking(). Here, Bicycle is a prototype. Now, we can create any number of bicycles using the prototype. And, all the bicycles will share the fields and methods of the prototype.

**Note:** We have used keywords private and public. These are known as access modifiers.

### **Java Objects**

An object is called an instance of a class. For example, suppose Bicycle is a class then MountainBicycle, SportsBicycle, TouringBicycle, etc can be considered as objects of the class.

## **Creating an Object in Java**

Here is how we can create an object of a class.

```
className object = new className();
// for Bicycle class
```

```
Bicycle sportsBicycle = new Bicycle();
Bicycle touringBicycle = new Bicycle();
```

Here, sportsBicycle and touringBicycle are the names of objects. We can use them to access fields and methods of the class. As you can see, we have created two objects of the class. We can create multiple objects of a single class in Java.

**Note:** Fields and methods of a class are also called members of the class.

## **Access Members of a Class**

We can use the name of objects along with the **. operator** to access members of a class. For example,

```
class Bicycle {
```

```
// field of class
int gear = 5;
// method of class
void braking() {
 ...
}
```

```
}
```

```
// create object
Bicycle sportsBicycle = new Bicycle();
// access field and method
sportsBicycle.gear;
```

In the above example, we have created a class named Bicycle. It includes a field named gear.

Here, we have created an object of Bicycle named sportsBicycle. We then use the object to access the field of the class.

## **Java Package**

A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

### **Advantage of Java Package**

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java

package mypack;

public class Simple
{
 public static void main(String args[]){
 System.out.println("Welcome to package");
 }
}
```

### How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

*javac -d directory javafilename*

Example:

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

To Compile: `javac -d . Simple.java`

To Run: `java mypack.Simple`

Output:

```
Welcome to package
```

### How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

#### 1) Using packagename.\*

If you use `package.*` then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.\*

```
//save by A.java
package pack;

public class A{
 public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
```

```
class B{
```

```
public static void main(String args[]){
 A obj = new A();
 obj.msg();
}
```

Output

```
Hello
```

## 2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java
```

```
package pack;
public class A{
 public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
```

```
package mypack;
import pack.A;
```

```
class B{
 public static void main(String args[]){
 A obj = new A();
 obj.msg();
 }
}
```

Output:

```
Hello
```

## 3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
 public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
class B{
 public static void main(String args[]){
 pack.A obj = new pack.A();//using fully qualified name
 obj.msg();
 }
}
```

Output:

```
Hello
```

## Methods

A method in Java or Java Method is a collection of statements that perform some specific tasks and return the result to the caller. A Java method can perform some specific tasks without returning anything. Methods in Java allow us to reuse the code without retyping the code.

### Method Declaration

In general, method declarations have **six components**:

- Modifier: It defines the access type of the method i.e. from where it can be accessed in your application. In Java, there 4 types of access specifiers.
  - public: It is accessible in all classes in your application.
  - protected: It is accessible within the class in which it is defined and in its subclass/es
  - private: It is accessible only within the class in which it is defined.
  - default: It is declared/defined without using any modifier. It is accessible within the same class and package within which its class is defined.
- The return type: The data type of the value returned by the method or void if does not return a value.
- Method Name: the rules for field names apply to method names as well, but the convention is a little different.
- Parameter list: Comma-separated list of the input parameters is defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses () .
- Exception list: The exceptions you expect by the method can throw; you can specify these exception(s).
- Method body: it is enclosed between braces. The code you need to be executed to perform your intended operations.

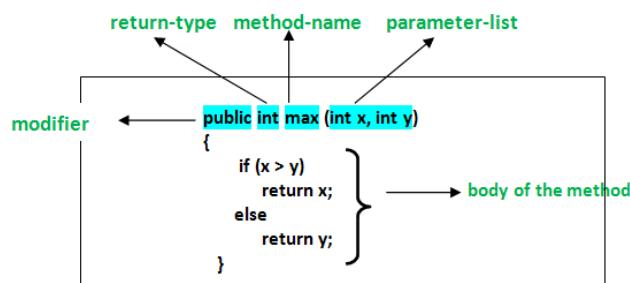


Image: Method Declaration

## Calling a Method

```
// calls the method
addNumbers();
```

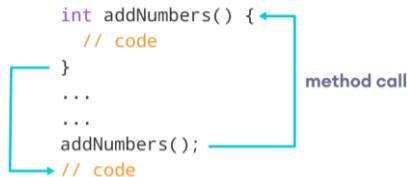


Image: Calling a Method

## Example

```
class Main {

 // create a method
 public int addNumbers(int a, int b) {
 int sum = a + b;
 // return value
 return sum;
 }

 public static void main(String[] args) {

 int num1 = 25;
 int num2 = 15;

 // create an object of Main
 Main obj = new Main();
 // calling method
 int result = obj.addNumbers(num1, num2);
 System.out.println("Sum is: " + result);
 }
}
```

## Output

Sum is: 40

In the above example, we have created a method named `addNumbers()`. The method takes two parameters `a` and `b`. Notice the line,

```
int result = obj.addNumbers(num1, num2);
```

Here, we have called the method by passing two arguments `num1` and `num2`. Since the method is returning some value, we have stored the value in the `result` variable.

**Note:** The method is not static. Hence, we are calling the method using the object of the class.

## Method Return Type

A Java method may or may not return a value to the function call. We use the `return` statement to return any value. For example,

```
int addNumbers() {
```

```
...
return sum;
}
```

Here, we are returning the variable sum. Since the return type of the function is int. The sum variable should be of int type. Otherwise, it will generate an error.

### Example:

```
class Main {
// create a method
public static int square(int num) {
// return statement
return num * num;
}
public static void main(String[] args) {
int result;
// call the method
// store returned value to result
result = square(10);

System.out.println("Squared value of 10 is: " + result);
}
}
```

### Output:

Squared value of 10 is: 100

```
int square(int num) {
 return num * num;
}
...
...
result = square(10);
// code
```

Image: Method Return Type

In the above program, we have created a method named square (). The method takes a number as its parameter and returns the square of the number.

Here, we have mentioned the return type of the method as int. Hence, the method should always return an integer value.

### Types of Methods in Java

There are two types of methods in Java:

**1. Predefined Method:** In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point.

### Example:

```
public class Main {
public static void main(String[] args) {
```

```

// using the sqrt() method
System.out.print("Square root of 4 is: " + Math.sqrt(4));
}
}

```

**Output:**

Square root of 4 is: 2.0

**2. User-defined Method:** The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

**Example :**

```

import java.util.Scanner;
public class EvenOdd
{
 public static void main (String args[])
 {
 //creating Scanner class object
 Scanner scan=new Scanner(System.in);
 System.out.print("Enter the number: ");
 //reading value from user
 int num=scan.nextInt();
 //method calling
 findEvenOdd(num);
 }
 //user defined method
 public static void findEvenOdd(int num)
 {
 //method body
 if(num%2==0)
 System.out.println(num+" is even");
 else
 System.out.println(num+" is odd");
 }
}

```

**Output:**

Enter the number: 12

12 is even

Output 2:

Enter the number: 99

99 is odd

### Static Method

The static keyword is used to construct methods that will exist regardless of whether or not any instances of the class are generated. Any method that uses the static keyword is referred to as a static method.

### Features of static method:

- A static method in Java is a method that is part of a class rather than an instance of that class.
- Every instance of a class has access to the method.
- Static methods have access to class variables (static variables) without using the class's object (instance).
- Only static data may be accessed by a static method. It is unable to access data that is not static (instance variables).
- In both static and non-static methods, static methods can be accessed directly.

**Syntax:**

```
Access_modifier static void methodName()
{
 // Method body.
}
```

The name of the class can be used to invoke or access static methods.

**Syntax to call a static method:**

```
className.methodName();
```

**Example**

```
class JavaExample{
 static int i = 100;
 static String s = "book";
 //Static method
 static void display()
 {
 System.out.println("i:"+i);
 System.out.println("i:"+s);
 }

 //non-static method
 void funcn()
 {
 //Static method called in non-static method
 display();
 }
 //static method
 public static void main(String args[])
 {
 JavaExample obj = new JavaExample();
 //You need to have object to call this non-static method
 obj.funcn();

 //Static method called in another static method
 display();
 }
}
```

**Output:**

i:100

```
i:book
i:100
i:book
```

## Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method. Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a String called fname as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

```
public class Main {
 static void myMethod(String fname) {
 System.out.println(fname + " Refsnes");
 }

 public static void main(String[] args) {
 myMethod("Liam");
 myMethod("Jenny");
 myMethod("Anja");
 } }
```

## Output

```
Liam Refsnes
Jenny Refsnes
Anja Refsnes
```

**Note:** When a parameter is passed to the method, it is called an **argument**. So, from the example above: fname is a parameter, while Liam, Jenny and Anja are arguments.

## Multiple Parameters

The following example method takes two integer arguments and returns their sum.

```
public class Main {
 static int myMethod(int x, int y) {
 return x + y;
 }

 public static void main(String[] args) {
 int z = myMethod(5, 3);
 System.out.println(z);
 } }
```

## Output

```
8
```

## Constructors

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default. There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

### Rules for creating Java constructor

There are two rules defined for the constructor.

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

### Types of Java constructors

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor

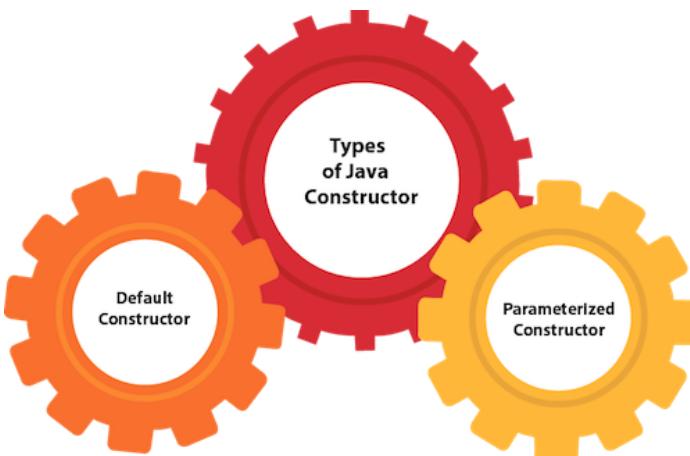


Image: Types of Constructors

### Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

#### Syntax:

```
<class_name>(){}
```

#### Example:

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
```

```
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

### Output:

```
Bike is created
```

### Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
 int id;
 String name;
 //creating a parameterized constructor
 Student4(int i,String n){
 id = i;
 name = n;
 }
 //method to display the values
 void display(){System.out.println(id+" "+name);}

 public static void main(String args[]){
 //creating objects and passing values
 Student4 s1 = new Student4(111,"Raja");
 Student4 s2 = new Student4(222,"Rani");
 //calling method to display the values of object
 s1.display();
 s2.display();
 }
}
```

### Output:

```
111 Raja
222 Rani
```

## Constructor Overloading

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

**Example:**

```
//Java program to overload constructors
class Student5{
 int id;
 String name;
 int age;
 //creating two arg constructor
 Student5(int i,String n){
 id = i;
 name = n;
 }
 //creating three arg constructor
 Student5(int i,String n,int a){
 id = i;
 name = n;
 age=a;
 }
 void display(){System.out.println(id+" "+name+" "+age);}

 public static void main(String args[]){
 Student5 s1 = new Student5(111,"Raja");
 Student5 s2 = new Student5(222,"Rani",25);
 s1.display();
 s2.display();
 }
}
```

**Output:**

```
111 Raja 0
222 Rani 25
```

**Difference between constructor and method**

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

## Strings

A string is an immutable object representing a sequence of characters in Java. The immutable property does not allow you to modify a single character of the string, you have to delete the whole string or make a new one. String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The `java.lang.String` class is used to create a string object.

### How to create a string object?

There are two ways to create String object:

- By string literal
- By new keyword

#### Method 1: Using a string literal

A string literal is the most common practice being followed to create a new string in Java. The first syntax provided below refers to creating a string using a string literal:

**String s=<value>**

The instances in the above syntax are:

- String is the keyword used to create string literals
- s is the string object's name
- the <value> is the sequence of characters

Whenever the string object is created using the string literal method, JVM matches the string(being created) in the existing list of strings (from string constant pool). If the string already exists, this method will not create a new string, it will refer to the already stored string.

**Java String literal is created by using double quotes.**

#### For Example:

`String s="welcome";`

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. **For example:**

`String s1="Welcome";`

String s2="Welcome";//It doesn't create a new instance

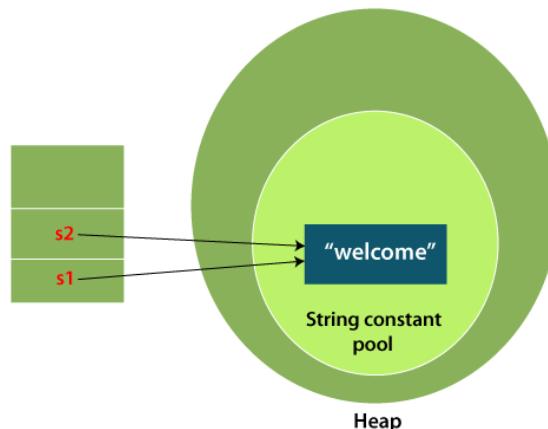


Image: String literal

Reference: <https://www.javatpoint.com/java-string>

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

**Note: String objects are stored in a special memory area known as the "string constant pool".**

## Method 2: Using the new operator

The following syntax can be followed to create a string in Java using the new keyword.

**String = new String("<value>")**

The new operator always creates a new object rather than referring to the already stored string. Thus, it is recommended to create a string using the string literal as this method optimizes the memory as well.

String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

## Example

```
public class StringExample{
 public static void main(String args[]){
 String s1="java";//creating string by Java string literal
 char ch[]={‘s’,’l’,’r’,’i’,’n’,’g’,’s’};
 String s2=new String(ch);//converting char array to string
 String s3=new String("example");//creating Java string by new keyword
 System.out.println(s1);
 System.out.println(s2);
 System.out.println(s3);
 }
}
```

## Output:

java  
strings  
example

## Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. In Java, String objects are immutable. Immutable simply means unmodifiable or unchangeable. Once String object is created its data or state can't be changed but a new String object is created.

### Example

```
class Testimmutablestring{
 public static void main(String args[]){
 String s="Sachin";
 s.concat(" Tendulkar");//concat() method appends the string at the end
 System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

### Output:

Sachin

Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as **immutable**.

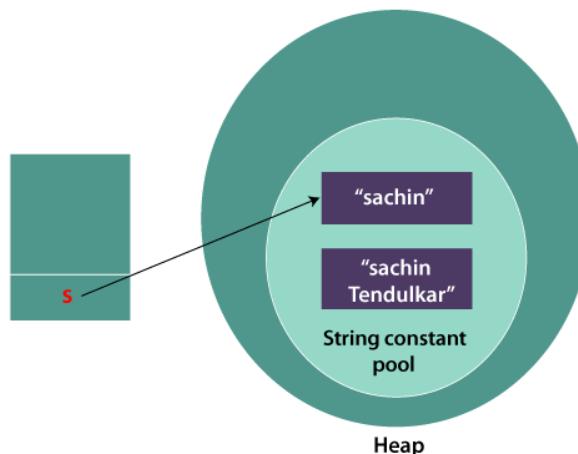


Image: immutable string

Reference: <https://www.javatpoint.com/immutable-string>

As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar". But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

```
class Testimmutablestring1{
 public static void main(String args[]){
 String s="Sachin";
```

```
s=s.concat(" Tendulkar");
System.out.println(s);
}
```

### Output:

```
Sachin Tendulkar
```

In such a case, s points to the "Sachin Tendulkar". Please notice that still Sachin object is not modified.

### Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java. Following are some features of String which makes String objects immutable.

- **ClassLoader:** A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different. To avoid this kind of misinterpretation, String is immutable.
- **Thread Safe:** As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.
- **Security:** As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.
- **Heap Space:** The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

## String Methods

**length():** This method returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

### Syntax:

```
public int length()
```

### Example

```
public class Sample_String{
 public static void main(String[] args){
 String str_Sample = "RockStar";
 //Length of a String
 System.out.println("Length of String: " + str_Sample.length());}}
```

**Output:**

Length of String: 8
---------------------

**indexOf()**

The indexOf() method returns the position of the first occurrence of specified character(s) in a string

**Syntax:**

```
public int indexOf(String str)
public int indexOf(String str, int fromIndex)
public int indexOf(int char)
public int indexOf(int char, int fromIndex)
```

**Example**

```
public class Sample_String{

 public static void main(String[] args){//Character at position
String str_Sample = "RockStar";
System.out.println("Character at position 5: " + str_Sample.charAt(5));
//Index of a given character
System.out.println("Index of character 'S': " + str_Sample.indexOf('S'));
```

**Output:**

Character at position 5: t Index of character 'S': 4
---------------------------------------------------------

**charAt()**

The charAt() method returns the character at the specified index in a string. The index of the first character is 0, the second character is 1, and so on.

**Syntax:**

```
public char charAt(int index)
```

**Example**

```
public class Sample_String{
 public static void main(String[] args){//Character at position
String str_Sample = "RockStar";
System.out.println("Character at position 5: " + str_Sample.charAt(5));}
```

**Output:**

Character at position 5: t
----------------------------

**CompareTo()** : The compareTo() method compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The method returns 0 if the string is equal to the other string.

Use the method “compareTo” and specify the String that you would like to compare. Use “compareTolgnoreCase” in case you don’t want the result to be case sensitive. The result will have the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

#### Syntax:

```
public int compareTo(String string2)
public int compareTo(Object object)
```

#### Example

```
public class Sample_String{
 public static void main(String[] args){//Compare to a String
String str_Sample = "RockStar";
 System.out.println("Compare To 'ROCKSTAR': " +
str_Sample.compareTo("rockstar"));
 //Compare to - Ignore case
 System.out.println("Compare To 'ROCKSTAR' - Case Ignored: " +
str_Sample.compareTolgnoreCase("ROCKSTAR"));}}
```

#### Output:

```
Compare To 'ROCKSTAR': -32
Compare To 'ROCKSTAR' - Case Ignored: 0
```

**Contains(): The contains() method checks whether a string contains a sequence of characters. Returns true if the characters exist and false if not.**

Use the method “contains” and specify the characters you need to check.Returns true if and only if this string contains the specified sequence of char values.

#### Syntax:

```
public boolean contains(CharSequence chars)
```

#### Example

```
public class Sample_String{
 public static void main(String[] args){ //Check if String contains a sequence
String str_Sample = "RockStar";
 System.out.println("Contains sequence 'tar': " + str_Sample.contains("tar"));}}
```

#### Output:

```
Contains sequence 'tar': true
```

**startsWith() & endsWith(): The endsWith() method checks whether a string ends with the specified character(s).**

Returns true if the character sequence represented by the argument is a suffix of the character sequence represented by this object.

**Syntax:**

```
public boolean endsWith(String chars)
public boolean startsWith(String chars)
```

**Example**

```
public class Sample_String{
 public static void main(String[] args){ //Check if ends with a particular sequence
String str_Sample = "RockStar";
 System.out.println("EndsWith character 'r': " + str_Sample.endsWith("r"));}}
```

**Output:**

EndsWith character 'r': true

**replaceAll() & replaceFirst(): Java String Replace, replaceAll and replaceFirst methods. You can specify the part of the String you want to replace and the replacement String in the arguments.**

**Example**

```
public class Sample_String{
 public static void main(String[] args){//Replace Rock with the word Duke
String str_Sample = "RockStar";
System.out.println("Replace 'Rock' with 'Duke': " + str_Sample.replace("Rock",
"Duke"));}}
```

**Output:**

Replace 'Rock' with 'Duke': DukeStar

**toLowerCase() & Java touppercase(): Just use the “toLowerCase()” or “ToUpperCase()” methods against the Strings that need to be converted.**

**Syntax:**

```
public String toLowerCase()
public String toUpperCase()
```

**Example**

```
public class Sample_String{
 public static void main(String[] args){//Convert to LowerCase
String str_Sample = "RockStar";
System.out.println("Convert to LowerCase: " + str_Sample.toLowerCase());}
```

```
//Convert to Uppercase
System.out.println("Convert to Uppercase: " + str_Sample.toUpperCase());}}
```

### Output:

Convert to LowerCase: rockstar  
 Convert to Uppercase: ROCKSTAR

### Access Modifiers in Java

Access modifiers are keywords in Java that are used to set accessibility. An access modifier restricts the access of a class, constructor, data member and method in another class. Java language has four access modifiers to control access level for classes and its members.

- Default: Default has scope only inside the same package
- Public: Public has scope that is visible everywhere
- Protected: Protected has scope within the package and all sub classes
- Private: Private has scope only within the classes

Java also supports many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient etc.

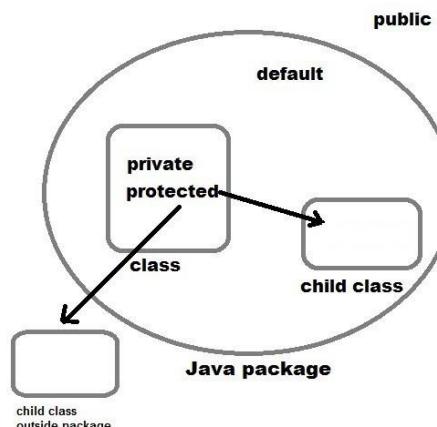


Image: Access modifiers

Reference: [https://miro.medium.com/max/605/0\\*4k3XjYwtO7k5FHuP.jpg](https://miro.medium.com/max/605/0*4k3XjYwtO7k5FHuP.jpg)

Access Modifier	Within Class	Within Package	Same Package by subclasses	Outside Package by subclasses	Global
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Image: Access modifiers

Reference: [https://1.bp.blogspot.com/-GCbzAxJ3\\_t8/XeqabcPeFCI/AAAAAAAABRY/LTJz83SB0zw9Ur7SNaEb2wMq3-QiEmuIACLcBGAsYHQ/s640/Access\\_Modifier.png](https://1.bp.blogspot.com/-GCbzAxJ3_t8/XeqabcPeFCI/AAAAAAAABRY/LTJz83SB0zw9Ur7SNaEb2wMq3-QiEmuIACLcBGAsYHQ/s640/Access_Modifier.png)

## **Default Access Modifier**

If we don't specify any access modifier then it is treated as default modifier. It is used to set accessibility within the package. It means we cannot access its method or class from outside the package. It is also known as package accessibility modifier.

### **Example:**

In this example, we created a Demo class inside the package1 and another class Test by which we are accessing show() method of Demo class. We did not mentioned access modifier for the show() method that's why it is not accessible and reports an error during compile time.

```
//Demo.java
package package1;
public class Demo {
 int a = 10;
 // default access modifier
 void show() {
 System.out.println(a);
 }
//Test.java
import package1.Demo;
public class Test {
 public static void main(String[] args) {
 Demo demo = new Demo();
 demo.show(); // compile error
 }
}
```

### **Output:**

The method show() from the type Demo is not visible

## **Public Access Modifier**

Public access modifier is used to set public accessibility to a variable, method or a class. Any variable or method which is declared as public can be accessible from anywhere in the application.

### **Example:**

Here, we have two class Demo and Test located in two different package. Now we want to access show method of Demo class from Test class. The method has public accessibility so it works fine. See the below example.

```
//Demo.java
package package1;
public class Demo {
 int a = 10;
 // public access modifier
 public void show() {
 System.out.println(a);
 }
}
```

```
 }
//Test.java
package package2;
import package1.Demo;
public class Test {
 public static void main(String[] args) {
 Demo demo = new Demo();
 demo.show();
 }
}
```

**Output:**

10

## Protected Access Modifier

Protected modifier protects the variable, method from accessible from outside the class. It is accessible within class, and in the child class (inheritance) whether child is located in the same package or some other package.

**Example:**

In this example, Test class is extended by Demo and called a protected method show() which is accessible now due to inheritance.

```
//Demo.java
package package1;
public class Demo {
 int a = 10;
 // public access modifier
 protected void show() {
 System.out.println(a);
 }
//Test.java
package package2;
import package1.Demo;
public class Test extends Demo{
 public static void main(String[] args) {
 Test test = new Test();
 test.show();
 }
}
```

**Output:**

10

## Private Access Modifier

Private modifier is most restricted modifier which allows accessibility within same class only. We can set this modifier to any variable, method or even constructor as well.

### **Example:**

In this example, we set private modifier to show() method and try to access that method from outside the class. Java does not allow to access it from outside the class.

```
//Demo.java
class Demo {
 int a = 10;
 private void show() {
 System.out.println(a);
 }
}
```

```
//Test.java
public class Test {
 public static void main(String[] args) {
 Demo demo = new Demo();
 demo.show(); // compile error
 }
}
```

### **Output:**

The method show() from the type Demo is not visible

## **Non-access Modifier**

Along with access modifiers, Java provides non-access modifiers as well. These modifiers are used to set special properties to the variable or method.

Non-access modifiers do not change the accessibility of variable or method, but they provide special properties to them. Java provides following non-access modifiers.

- Final
- Static
- Transient
- Synchronized
- Volatile

**Final Modifier :** Final modifier can be used with variable, method and class. if variable is declared final then we cannot change its value. If method is declared final then it cannot be overridden and if a class is declared final then we cannot inherit it.

**Static modifier :** Static Modifier is used to make field static. We can use it to declare static variable, method, class etc. static can be used to declare class level variable. If a method is declared static then we

**don't need to have object to access that. We can use static to create nested class.**

**Transient modifier :** When an instance variable is declared as transient, then its value doesn't persist when an object is serialized.

**Synchronized modifier :** When a method is synchronized it can be accessed by only one thread at a time. We will discuss it in detail in Thread.

**Volatile modifier :** Volatile modifier tells to the compiler that the volatile variable can be changed unexpectedly by other parts of a program. Volatile variables are used in case of multi-threading program. volatile keyword cannot be used with a method or a class. It can be only used with a variable.

### Java this keyword

In Java, this is a keyword which is used to refer current object of a class. we can it to refer any member of the class. It means we can access any instance variable and method by using this keyword. The main purpose of using this keyword is to solve the confusion when we have same variable name for instance and local variables.

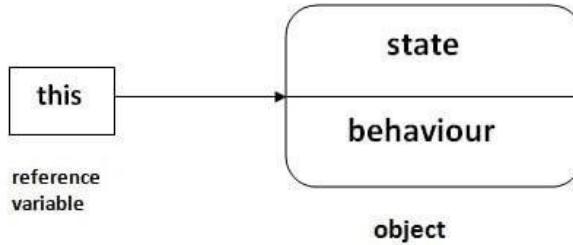


Image: this keyword

Reference: [https://miro.medium.com/max/605/0\\*4k3XjYwtO7k5FHuP.jpg](https://miro.medium.com/max/605/0*4k3XjYwtO7k5FHuP.jpg)

We can use this keyword for the following purpose.

- this keyword is used to refer to current object.
- this is always a reference to the object on which method was invoked.
- this can be used to invoke current class constructor.
- this can be passed as an argument to another method.

### Example:

In this example, we have three instance variables and a constructor that have three parameters with same name as instance variables. Now, we will use this to assign values of parameters to instance variables.

```
class Demo
{
 Double width, height, depth;
 Demo (double w, double h, double d)
 {
 this.width = w;
 this.height = h;
 this.depth = d;
 }
 public static void main(String[] args) {
 Demo d = new Demo(10,20,30);
 System.out.println("width = "+d.width);
 System.out.println("height = "+d.height);
 System.out.println("depth = "+d.depth);
 }
}
```

### **Output:**

```
width = 10.0
height = 20.0
depth = 30.0
```

Here this is used to initialize member of current object. Such as, this.width refers to the variable of the current object and width only refers to the parameter received in the constructor i.e the argument passed while calling the constructor.

## **Calling Constructor using this keyword**

We can call a constructor from inside another function by using this keyword

### **Example:**

In this example, we are calling a parameterized constructor from the non-parameterized constructor using this keyword along with argument.

```
class Demo
{
 Demo ()
 {
 // Calling constructor
 this("Hi");
 }

 Demo(String str){
 System.out.println(str);
```

```
}
```

```
public static void main(String[] args) {
```

```
 Demo d = new Demo(); }
```

```
}
```

#### **Output:**

Hi

### **Accessing Method using this keyword**

This is another use of this keyword that allows to access method. We can access method using object reference too but if we want to use implicit object provided by Java then use this keyword.

#### **Example:**

In this example, we are accessing getName() method using this and it works fine as works with object reference. See the below example

```
class Demo
```

```
{
```

```
 public void getName()
```

```
 {
```

```
 System.out.println("Hi");
```

```
 }
```

```
 public void display()
```

```
 {
```

```
 this.getName();
```

```
 }
```

```
 public static void main(String[] args)
```

```
 {
```

```
 Demo d = new Demo();
```

```
 d.display();
```

```
 }
```

```
}
```

#### **Output:**

Hi

### **Return Current Object from a Method**

In such scenario, where we want to return current object from a method then we can use this to solve this problem.

#### **Example:**

In this example, we created a method display that returns the object of Demo class. To return the object, we used this keyword and stored the returned object into Demo type reference variable. We used that returned object to call getName() method and it works fine.

```
class Demo
```

```
{
```

```
 public void getName()
```

```
 {
```

```
 System.out.println("Hi");
```

```

 }
 public Demo display()
 {
 // return current object
 return this;
 }
 public static void main(String[] args) {
 Demo d = new Demo();
 Demo d1 = d.display();
 d1.getName();
 }
}

```

### **Output:**

Hi

### **Static Keyword**

The static keyword in Java is used for memory management mainly. It is a keyword that is used to share the same variable or method of a given class. Basically, static is used for a constant variable or a method that is the same for every instance of a class. The static keyword can be used with class, variable, method, and block. Static members belong to the class instead of a specific instance, this means if you make a member static, you can access it without an object. If we want to access class members without creating an instance of the class, we need to declare the class members static. In Java programming language, static keyword is a non-access modifier and can be used for the following:

- Static Block
- Static Variable (also known as class variable)
- Static Method (also known as class method)
- Static Classes (Nested Classes)

### **Static Block in Java**

It is used to initialize the static data member. It is executed before the main method at the time of class loading. If you need to do the computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded. We can't access non-static variables in the static block. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

### **Syntax:**

```

static{
 //variable initialization
}

```

### **Example:**

```

package Demo;
import java.util.*;
public class StaticBlockDemo
{
 //static variable
 static int j = 10;
}

```

```

static int n;

//static block
static
{
 System.out.println ("Static block initialized.");
 n = j * 8;
}
public static void main (String[]args)
{
 System.out.println ("Inside main method");
 System.out.println ("Value of j : " + j);
 System.out.println ("Value of n : " + n);
}
}

```

### **Output:**

Static block initialized  
 Inside main method  
 Value of j: 10  
 Value of n: 80

### **Static Variable in Java**

If you declare any variable as static, it is known as a static variable. When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. It doesn't matter how many times we initialize a class; there will always be only one copy of a static field belonging to it. The value of this static field will be shared across all objects of either same or any different classes. Static variables are, essentially, global variables. A static variable is common to all the instances (or objects) of the class because it is a class-level variable. Local variables cannot be declared static. If the static variable is not private, we can access it with `ClassName.variableName`

### **Syntax:**

`static datatype variable-name`

### **Example:**

```

package Demo;
class Counter {
 static int count = 0;
 Counter(){
 count++;
 }
 public void getCount() {
 System.out.printf("Value of counter: %d \n", count);
 }
 public static void main(String args[]) {
 Counter c1 = new Counter(); //count incremented to 1
 c1.getCount();
 Counter c2 = new Counter(); //count incremented to 2
 }
}

```

```

 c2.getCount();
 Counter c3 = new Counter(); //count incremented to 3
 c3.getCount();
 }
}

```

**Output:**

Value of counter: 1  
 Value of counter: 2  
 Value of counter: 3

**Note:**

- We can create static variables at the class level only.
- Static block and static variables are executed in the order they are present in a program.
- It makes your program memory efficient (i.e., it saves memory).

## Static Methods in Java

A method declared with the static keyword. A static method can access only static variables of a class and invoke only static methods of the class. Usually, static methods are utility methods that we want to expose to be used by other classes without the need of creating an instance. One of the basic rules of working with static methods is that you can't access a non-static method or field from a static method because static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.

**Note:** The best-known static method is the main, which is called by the Java runtime to start an application. The main method must be static, which means that applications run in a static context by default.

**Syntax:**

static return\_type method\_name();

**Example:**

```

class StaticTest
{
 // non-static method
 int multiply (int a, int b)
 {
 return a * b;
 }
 // static method
 static int add (int a, int b)
 {
 return a + b;
 }
}
public class StaticMethodDemo

```

```

{
 public static void main (String[]args)
 {
 // create an instance of the StaticTest class
 StaticTest st = new StaticTest ();
 // call the nonstatic method
 System.out.println (" 5 * 5 = " + st.multiply (5, 5));
 // call the static method
 System.out.println (" 5 + 3 = " + StaticTest.add (5, 3));
 }
}

```

### **Output:**

5\*5=25  
5+3=8

### **Restrictions for the static method in Java**

There are two main restrictions for the static method. They are:

- The static method cannot use non-static data members or call the non-static method directly.
- this and super cannot be used in a static context.

### **Why is the Java main method static?**

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first and then calls the main() method which will lead to the problem of extra memory allocation.

### **Why it is not required to create an instance of a class?**

Usually, static methods are utility methods that we want to expose to be used by other classes without the need of creating an instance. The static keyword is used to create methods that will exist independently of any instances created for the class. As static methods take all the data from parameters and compute something from those parameters, with no reference to variables. And Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method. Therefore, the Static Method doesn't require instance creation, so it's generally faster and provides better performance. That's why utility class methods in Wrapper classes, System class, and Collections class are all static methods.

**Note:** The best-known static method is the main, which is called by the Java runtime to start an application. The main method must be static, which means that applications run in a static context by default.

### **When to create static methods in Java?**

It's possible to write fluent code when static imports are used. When your method only depends on its parameters, the object state has no effect on the method behavior. Then you can create the method as static.

## **Static Class in Java**

A class can be made static only if it is a nested class. Java programming language allows us to create a class within a class. It provides a compelling way of grouping elements that are only going to be used in one place, this helps to keep our code more organized and readable.

**Basically, nested classes are of two types:**

1. **Static Nested Classes:** nested classes that are declared static are called static nested classes
2. **Not-static Nested Classes:** nested classes that are non-static are called inner classes or non-static nested classes

**Syntax:**

```
class OuterClass {
 static class InnerClass {
 //code
 }
}
```

**Example:**

```
class StaticClassDemo
{
 //Outer Class
 static String message = "Hello World!";
 static class InnerClass
 {
 //Inner Class
 static void getMessage ()
 {
 System.out.println (message);
 }
 }
 public static void main (String args[])
 {
 StaticClassDemo.InnerClass.getMessage ();
 }
}
```

**Output:**

Hello World!

## Encapsulation

### What is Encapsulation in Java?

The meaning of Encapsulation is to make sure that “sensitive” data is hidden from users. Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. If a data member is private, it means it can only be accessed within the same class. No outside class can access private data members (variable) of other classes.

## **Need for Encapsulation in Java:**

- Better control of class attributes and methods.
- Encapsulation helps us to keep related fields and methods together, which makes our code cleaner and easy to read.
- Helps us to achieve a loose couple.
- An object exposes its behavior by means of public methods or functions.
- Increased security of data.

## **What is Data Hiding in Java?**

Data hiding was introduced as part of the OOP methodology, in which a program is segregated into objects with specific data and functions. This technique enhances a programmer's ability to create classes with unique data sets and functions, avoiding unnecessary penetration from other program classes. Encapsulation refers to the bundling of related fields and methods together. This allows us to achieve data hiding. Encapsulation in itself is not data hiding. Please have a look at the following image for a better understanding.

## **Difference Between Data Hiding and Encapsulation in Java:**

- Data hiding only hides class data components, whereas data encapsulation hides class data parts and private methods.
- Data hiding focuses more on data security and data encapsulation focuses more on hiding the complexity of the system.

## **How to achieve Encapsulation in Java?**

To achieve encapsulation in Java, you need to:

- declare class variables/attributes as private
- provide public getter and setter methods to access and update the value of a private variable

## **Getter and Setter Methods in Java:**

- The get method returns the variable value, and the set method sets the value.
- If a data member is declared "private", then it can only be accessed within the same class. No outside class can access data members of that class. If you need to access these variables, you have to use public "getter" and "setter" methods.
- Getter and Setter's methods are used to create, modify, delete, and view the values of the variables.
- The syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in upper case.
- In Java getters and setters are completely ordinary functions. The only thing that makes them getters or setters is a convention.
- A getter for demo is called getDemo and the setter is called setDemo .
- It should have a statement to assign the argument value to the corresponding variable.

## **Example:**

```
public class EncapsulationDemo
{
 // private variables declared
 // these can only be accessed by
 // public methods of class
 private String Name;
```

```

private int RollNo;
private int Age;
// get method for age to access
// private variable Age
public int getAge ()
{
 return Age;
}

// get method for name to access
// private variable Name
public String getName ()
{
 return Name;
}
// get method for roll to access
// private variable RollNo
public int getRoll ()
{
 return RollNo;
}
// set method for age to access
// private variable Age
public void setAge (int newAge)
{
 Age = newAge;
}
// set method for name to access
// private variable Name
public void setName (String newName)
{
 Name = newName;
}
// set method for roll to access
// private variable RollNo
public void setRoll (int newRollNo)
{
 RollNo = newRollNo;
}
public static void main (String[]args)
{
 EncapsulationDemo obj = new EncapsulationDemo ();
 // setting values of the variables
 obj.setName ("Harsh");
 obj.setAge (19);
 obj.setRoll (51);
 // Displaying values of the variables
 System.out.println ("Student's Name: " + obj.getName ());
 System.out.println ("Student's Age: " + obj.getAge ());
 System.out.println ("Student's RollNo: " + obj.getRoll ());
}

```

```
}
```

### Output:

```
Student's Name: Harry
```

```
Student's Age: 19
```

```
Student's RollNo: 51
```

### Inheritance with Types

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

### Why use inheritance in java?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

### Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

### Syntax:

```
class Subclass-name extends Superclass-name
{
 //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

### Example:

```
class Employee { float salary=40000; }
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
 Programmer p=new Programmer();
 System.out.println("Programmer salary is:"+p.salary);
 System.out.println("Bonus of Programmer is:"+p.bonus);
```

```
}
```

### Output:

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

### Types of inheritance in java

- Single
- Multiple
- Multilevel
- Hybrid
- Hierarchical

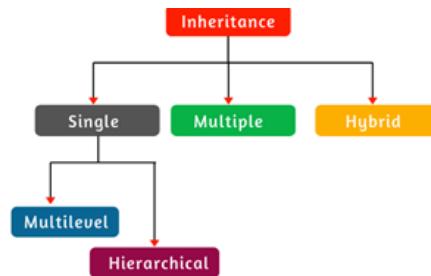


Image: Inheritance with Types

Reference: [https://www.scientecheeasy.com/2020/07/types-of-inheritance-in-java.html/](https://www.scientecheasy.com/2020/07/types-of-inheritance-in-java.html/)

- **Single Inheritance:** When a derived class or subclass inherits from only one base or superclass then it is single inheritance.
- **Multilevel Inheritance:** In Multilevel Inheritance, we have more than one level wherein a class inherits from a base class and the derived class in turn is inherited by another class.
- **Hierarchical Inheritance:** An inheritance hierarchy is formed in this type of inheritance when a superclass is inherited by more than one class.
- **Multiple Inheritance:** Multiple inheritance is the one in which a class can inherit properties and behavior from more than one parent.
- **Hybrid Inheritance:** When one or more types of inheritance are combined, then it becomes a hybrid inheritance.

**Note: Multiple inheritance is not supported in Java through class.**

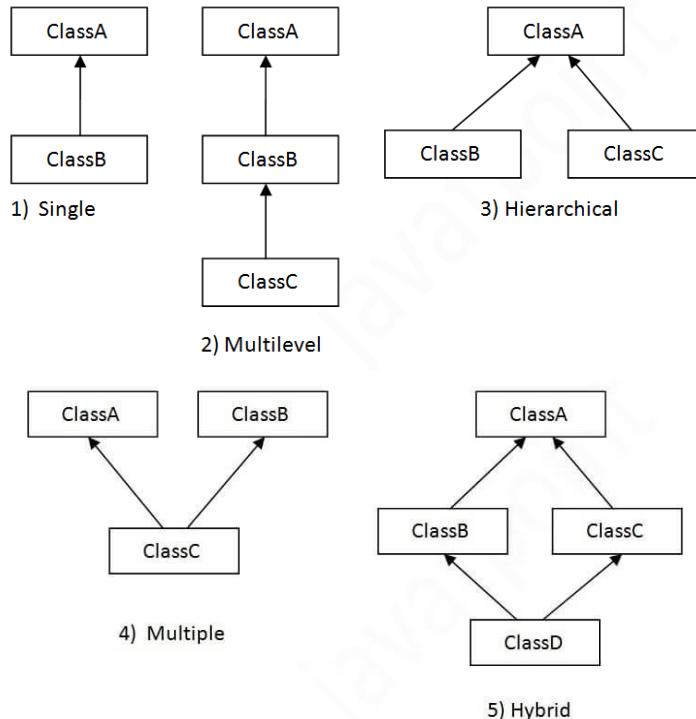


Image: Inheritance with Types  
 Reference: <https://www.javatpoint.com/inheritance-in-java>

## Single Inheritance

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

### Example:

```
//TestInheritance.java

class Animal{
void eat(){System.out.println("eating...");}
}

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}
}
```

**Output:**

```
barking...
eating...
```

**Multilevel Inheritance**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

**Example:**

```
//TestInheritance2.java

class Animal{
void eat(){System.out.println("eating...");}
}

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}

class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
```

**Output:**

```
weeping...
barking...
eating...
```

**Hierarchical Inheritance**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

**Example:**

```
//TestInheritance3.java

class Animal{
void eat(){System.out.println("eating...");}
}
```

```

class Dog extends Animal{
void bark(){System.out.println("barking...");}
}

class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}

class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark(); //C.T.Error
}
}

```

### **Output:**

```

meowing...
eating...

```

### **Why multiple inheritance is not supported in java?**

In java, multiple inheritance is not supported because of ambiguity problem. We can take the below example where we have two classes Class1 and Class2 which have same method display(). If multiple inheritance is possible than Test class can inherit data members (properties) and methods (behaviour) of both Class1 and Class2 classes. Now, Test class have two display() methods inherited from Class1 and Class2. Problem occurs in method call, when display() method will be called with Test class object which method will be called, will it be of Class1 or Class2. This is ambiguity problem because of which multiple inheritance is not supported in java.

### **Example:**

```

class Class1{
 public void display(){
 System.out.println("Display method inside Class1.");
 }
}

class Class2{
 public void display(){
 System.out.println("Display method inside Class2.");
 }
}

//let multiple inheritance is possible.
public class Test extends Class1, Class2{
 public static void main(String args[]){
 Test obj = new Test();
 }
}

```

```
//Ambiguity problem in method call which class display() method will be called.
 obj.display(); }
}
```

#### **Output:**

```
Exception in thread "main" java.lang.Error:
Unresolved compilation problem:
```

### **Method Overloading & Overriding**

Method overloading in Java means having two or more methods (or functions) in a class with the same name and different arguments (or parameters). It can be with a different number of arguments or different data types of arguments.

#### **For instance:**

- void function1(double a) { ... }
- void function1(int a, int b, double c) { ... }
- float function1(float a) { ... }
- double function1(int a, float b) { ... }

In the above example, function1() is overloaded using a different number of parameters and different data types of parameters.

#### **Example:**

```
class Method_Overloading {
 double figure(double l, int b) //two parameters with double type
 {
 return (l*b);
 }

 float figure(int s) //one parameter with float return type
 {
 return (s*s);
 }

 public static void main(String[] args) {
 Method_Overloading obj = new Method_Overloading();
 System.out.println("Area of Rectangle: " +obj.figure(5.55, 6));
 System.out.println("Area of Square: " +obj.figure(3));
 }
}
```

#### **Output:**

```
Area of Rectangle: 33.3
Area of Square: 9.0
```

### **By changing the Number of Parameters**

A method can differ with the number of parameters passed. The below example shows how we can implement method overloading with the different number of parameters in method declaration and definition.

**Example:**

```
class Method_Overloading {
 //Method Overloading by changing the number of arguments (or parameters)

 //Method 1
 double figure(double l, double b) //two arguments or parameters
 {
 return (l*b);
 }
 double figure(double s) //one argument or parameter
 {
 return (s*s);
 }

 //Method 2
 public static void main(String[] args) {
 Method_Overloading obj = new Method_Overloading();
 System.out.println("Area of Rectangle: " +obj.figure(5.55, 6.78));
 System.out.println("Area of Square: " +obj.figure(3.45));
 }
}
```

**Output:**

```
Area of Rectangle: 37.629
Area of Square: 11.902500000000002
```

**Changing the Data Type of Parameters**

Another way to do method overloading is by changing the data types of method parameters. The below example shows how we can implement method overloading with the different data types of parameters in method declaration and definition.

**Example:**

```
class Method_Overloading {
 //Method Overloading by changing the data type of arguments (or parameters)
 double figure(double l, double b) //method 1
 {
 return (l*b); //returns area of rectangle
 }
 double figure(int b, int h) //method 2
 {
 return ((b*h)/2); //returns area of right triangle
 }
 double figure(int b, double h) //method 3
 {
 return (b*h); //returns area of parallelogram
 }
}
```

```

 }

 public static void main(String[] args) {
 Method_Overloading obj = new Method_Overloading();
 System.out.println("Area of Rectangle: " +obj.figure(5.55, 6.78));
 System.out.println("Area of Right Triangle: " +obj.figure(3,5));
 System.out.println("Area of Parallelogram: " +obj.figure(4,6.3));
 }
}

```

### **Output:**

Area of Rectangle : 37.629  
 Area of Right Traingle : 7.0  
 Area of Parallelogram : 25.2

### **Can we overload main() in Java?**

Yes, evidently main() method can also be overloaded. The below example shows a simple implementation to overload the main() method with a different set of parameters.

### **Example:**

```

class Main_Overloading {

 //main 1
 public static void main(String[] args)
 {
 System.out.println("Overloading from main 1");
 Main_Overloading.main("User!");
 }

 //main 2
 public static void main(String arg1)
 {
 System.out.println("Hello, " + arg1 + " Overloading main 2");
 Main_Overloading.main("Are you learning", " Main Method
overloading!");
 }

 //main 3
 public static void main(String arg1, String arg2)
 {
 System.out.println("Overloading main 3");
 System.out.println("Hi, " + arg1 + ", " + arg2);
 }
}

```

### **Output:**

Overloading from main 1

```
Hello, User! Overloading main 2
Overloading main 3
Hi, Are you learning, Main Method overloading!
```

### Method Overriding

Method Overriding is redefining the method from the superclass into the subclass by adding new functionality or code. In Java, using the overriding concept superclass provides the freedom to the subclass for writing their own code instead of binding on superclass behavior or code.

### Usage of Method Overriding in Java:

Method overriding is used to provide the specific implementation of a method that is already provided by its superclass. Method overriding is used for runtime polymorphism.

### Rules for Method Overriding in Java:

- The method name must be the same.
- Method parameter types must be the same.
- The method return type must be the same.

### Example:

```
class connection
{
 void connect ()
 {
 }
}
class oracleconnection extends connection
{
 void connect ()
 {
 System.out.println ("Connected to Oracle");
 }
}
class mysqlconnection extends connection
{
 void connect ()
 {
 System.out.println ("Connected to MySQL");
 }
}
class Overriding
{
 public static void main (String args[])
 {
 oracleconnection a1 = new oracleconnection ();
 mysqlconnection b1 = new mysqlconnection ();
 a1.connect ();
 b1.connect ();
 }
}
```

```
}
```

### Output:

```
Connected to Oracle
Connected to MySQL
```

### Dynamic Binding in Java:

When the type of the object is determined at run time, it is known as dynamic binding. In Dynamic binding compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.

### Example:

```
public class Main
{
 public static class superclass
 {
 void print ()
 {
 //print in superclass
 System.out.println ("Hi!");
 }
 }
 public static class subclass extends superclass
 {
 @Override void print ()
 {
 //print in subclass
 System.out.println ("Hello!");
 }
 }
 public static void main (String[]args)
 {
 superclass A = new superclass ();
 superclass B = new subclass ();
 A.print ();
 B.print ();
 }
}
```

### Output:

```
Hi!
Hello!
```

### Abstract Class

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

## **Abstraction in Java**

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

## **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

## **Abstract class in Java**

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

**Note:** A method that does not have a body is called an abstract method and the class that is declared by using the abstract keyword is called an abstract class. If a class contains an abstract method, then it must be declared as abstract.

## **Abstract Method in Java**

A method that does not have a body is called an abstract method in Java. It is declared with the modifier abstract. The following are the properties of the java abstract method,

- It can only be used in an abstract class, and it does not have a body.
- An abstract method contains a method signature, but no method body.
- An abstract method is a method that is declared without implementation.
- Instead of curly braces, an abstract method will have a semicolon (;) at the end.
- A method-defined abstract must always be redefined in the subclass, thus making overriding compulsory OR either making the subclass itself abstract.

## **Rules of Abstract Class and Abstract Methods in Java:**

**Rule1:** If the method does not have a body it should be declared as abstract using the abstract modifier else it leads to CE: "missing method body or declared abstract"

```
public class Example
{
 void m1(); //CE: missing method body or declared abstract
}
CE: missing method body or declared abstract
```

**Rule2:** If a class has an abstract method, it should be declared as abstract by using the keyword abstract else it leads to CE: class is not abstract and does not override the abstract method in the class.

```
public class Example
{
 abstract void m1();
```

CE: Example is not abstract and does not override abstract method m1() in Example.  
The correct syntax is given below.

```
abstract class Example
{
 abstract void m1();
```

**Rule3:** If a class is declared as abstract it cannot be instantiated violation leads to compile-time Error.

```
abstract class Example
{
 abstract void m1();
 public static void main(String args[])
 {
 Example e = new Example();
 }
}
```

CE: Example is abstract, cannot be instantiated

**Rule4:** The subclass of an abstract class should override all abstract methods or it should be declared as abstract else it leads to CE:

```
abstract class Example
{
 abstract void m1();
 abstract void m2();
}
class Sample extends Example
{
 void m1()
 {
 System.out.println("m1 method");
 }
}
```

#### **Example:**

```
abstract class Bank
{
 abstract int getRateOfInterest ();
```

```
class SBI extends Bank
```

```

{
 int getRateOfInterest ()
 {
 return 7;
 }
}

class PNB extends Bank
{
 int getRateOfInterest ()
 {
 return 8;
 }
}

public class Main
{
 public static void main (String[]args)
 {
 Bank b;
 b = new SBI ();
 System.out.println ("SBI Rate of Interest is: " + b.getRateOfInterest () + "%");
 b = new PNB ();
 System.out.println ("PNB Rate of Interest is: " + b.getRateOfInterest () + "%");
 }
}

```

### **Output:**

SBI Rate of Interest is: 7%  
 PNB Rate of Interest is: 8%

### **Interfaces**

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class.

Since Java 8, we can have default and static methods in an interface.

Since Java 9, we can have private methods in an interface.

### **Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

## **How to declare an interface?**

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

### **Syntax:**

```
interface <interface_name>{
 // declare constant fields
 // declare methods that abstract
 // by default.
}
```

### **Example:**

```
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

### **Output:**

```
Hello
```

### **Example: Drawable**

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface

### **Example:**

```
//TestInterface1.java
//Interface declaration: by first user

interface Drawable{
void draw();
}

//Implementation: by second user

class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
```

```

}

class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}

//Using interface: by third user

class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g.
getDrawable()
d.draw();
}
}

```

**Output:**

drawing circle

**Example: Bank**

Let's see another example of java interface which provides the implementation of Bank interface.

```

//TestInterface2.java

interface Bank{
float rateOfInterest();
}

class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}

class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}

class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}
}

```

**Output:**

ROI: 9.15

**Multiple inheritance in Java by interface**

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

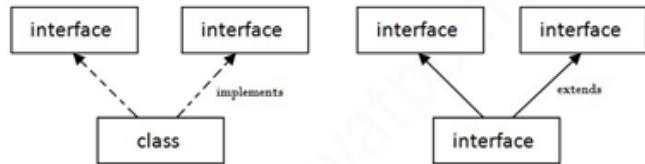


Image: Multiple inheritance in java interface

Reference: <https://www.javatpoint.com/interface-in-java>

### Example:

```
interface Printable{
 void print(); }
interface Showable{
 void show(); }
class A7 implements Printable,Showable{
 public void print(){ System.out.println("Hello"); }
 public void show(){ System.out.println("Welcome"); }
 public static void main(String args[]){
 A7 obj = new A7();
 obj.print();
 obj.show(); } }
```

### Output:

```
Hello
Welcome
```

## Exception Handling

### What are Errors?

The error signifies a situation that mostly happens due to the absence of system resources. The system crash and memory errors are an example of errors. It majorly occurs at runtime.

### What are Exceptions?

The exceptions are the issues that can appear at runtime and compile time. It majorly arises in the code or program authored by the developers. There are two types of exceptions: Checked exceptions and Unchecked exceptions.

We have the following two types of errors:

- Compile Time Error
- Run Time Error

### Compile Time Error

Errors that occur at the time of compilation of the program are called compile-time errors. Compile-time errors occurred because if we don't follow the java syntaxes properly, java programming rules properly, etc. Compile-time errors are identified by the java compiler. So, in simple words, we can say that compile-time errors occur due to a poor understanding of the programming language. These errors can be

identified by the programmer and can be rectified before the execution of the program only. So, these errors do not cause any harm to the program execution.

### Run Time Error

Errors that occur at the time of execution in the program are called runtime errors. Run-Time Errors are also called Exceptions. Exceptions may occur because programmer logic fails or JVM fails. Exceptions are identified by JVM.

**Note:** The Runtime errors are dangerous because whenever they occur in the program, the program terminates abnormally on the same line where the error gets occurred without executing the next line of code.

S.No	Errors	Exceptions
1.	The error indicates trouble that primarily occurs due to the scarcity of system resources.	The exceptions are the issues that can appear at runtime and compile time.
2.	It is not possible to recover from an error.	It is possible to recover from an exception.
3.	In java, all the errors are unchecked.	In java, the exceptions can be both checked and unchecked.
4.	The system in which the program is running is responsible for errors.	The code of the program is accountable for exceptions.
5.	They are described in the java.lang.Error package.	They are described in java.lang.Exception package

### Hierarchy of Java Exception classes

The `java.lang.Throwable` class is the root class of Java Exception hierarchy inherited by two subclasses: `Exception` and `Error`. The hierarchy of Java Exception classes is given below:

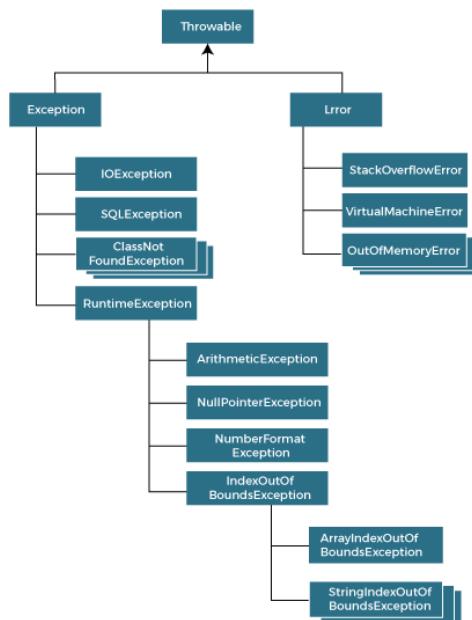


Image: Hierarchy of Java Exception

Reference: <https://static.javatpoint.com/core/images/hierarchy-of-exception-handling.png>

### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

- Checked Exception
- Unchecked Exception
- Error

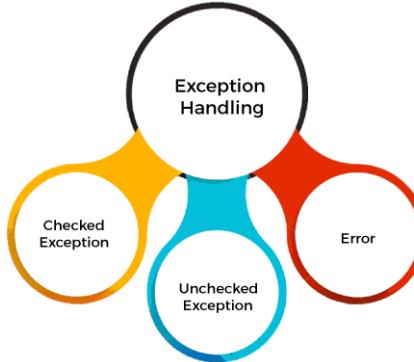


Image: Types of Java Exceptions

Reference: <https://static.javatpoint.com/core/images/hierarchy-of-exception-handling.png>

### Checked Exceptions in Java:

Exceptions that are identified at compilation time and occurred at runtime are called checked exceptions. These checked exceptions are also called Compile Time Exceptions. An exception said to be checked exception whose exception handling is mandatory as per the compiler. Example: IOException, ClassNotFoundException, CloneNotSupportedException, etc.

### Unchecked Exceptions in Java

Exceptions that are identified and occurred at run-time are called Unchecked Exceptions. These Unchecked Exceptions are also called Runtime Exceptions. An exception is said to be an unchecked exception whose exception handling is optional as per the compiler. Example: Arithmetic Exception, NumberFormatException, NoSuchElementException, etc.

**Note:** All child classes of Error and Runtime Exception classes are called the unchecked exception and the remaining classes are called checked exceptions.

In Java, Exception Handling can be done by using five Java keywords:

- Try
- Catch
- Finally
- Throw
- throws

#### try-block:

- The code which might raise exception must be enclosed within try-block
- try-block must be followed by either catch-block or finally-block, at the end
- If both present, it is still valid but the sequence of the try-catch-finally matters the most
- Otherwise, compile-time error will be thrown for invalid sequence
- The valid combination like try-catch block or try-catch-finally blocks must reside inside method

**Note:** code inside try-block must always be wrapped inside curly braces, even if it contains just one line of code; Otherwise, compile-time error will be thrown Compile-time Error : “Syntax error on token “”, Block expected after this token”

#### **catch-block:**

- Contains handling code for any exception raised from corresponding try-block and it must be enclosed within catch block
- catch-block takes one argument which should be of type Throwable or one of its sub-classes i.e.; class-name followed by a variable
- Variable contains exception information for exception raised from try-block
- Note: code inside catch-block must always be wrapped inside curly braces, even if it contains just one line of code;
- Otherwise, compile-time error will be thrown
- Compile-time Error: “Syntax error on token “”, Block expected after this token”

#### **finally-block:**

- finally block is used to perform clean-up activities or code clean-up like closing database connection & closing streams or file resources, etc
- finally block is always associated with try-catch block
- With finally-block, there can be 2 combinations
- One is try-block is followed by finally-block and other is try-catch-finally sequence
- The only other possible combination is try block followed by multiple catch block and one finally block at the end (this is case of multiple catch blocks)
- Advantage: The beauty of finally block is that, it is executed irrespective of whether exception is thrown or NOT (from try-block)
- Also, it gets executed whether respective exception is handled or NOT (inside catch-block)

**Note:** finally block won't get executed if JVM exits with System.exit() or due to some fatal error like code is interrupted or killed

#### **throw clause:**

- Sometimes, programmer can also throw/raise exception explicitly at runtime on the basis of some business condition
- To raise such exception explicitly during program execution, we need to use throw keyword

#### **Syntax: throw instanceOfThrowableType**

- Generally, throw keyword is used to throw user-defined exception or custom exception
- Although, it is perfectly valid to throw pre-defined exception or already defined exception in Java like IOException, NullPointerException, ArithmeticException, InterruptedException, ArrayIndexOutOfBoundsException, etc.

try {

```

// some valid Java statements
throw new RuntimeException();
}
catch(Throwable th) {

 // handle exception here
 // or re-throw caught exception
}

```

**throws keyword or throws clause:**

- throws keyword is used to declare the exception that might raise during program execution
- whenever exception might thrown from program, then programmer doesn't necessarily need to handle that exception using try-catch block instead simply declare that exception using throws clause next to method signature
- But this forces or tells the caller method to handle that exception; but again caller can handle that exception using try-catch block or re-declare those exception with throws clause

**Note:** use of throws clause doesn't necessarily mean that program will terminate normally rather it is the information to the caller to handle for normal termination

- Any number of exceptions can be specified using throws clause, but they are all need to be separated by commas (,)
- throws clause is applicable for methods & constructor but strictly not applicable to classes
- It is mainly used for checked exception, as unchecked exception by default propagated back to the caller (i.e.; up in the runtime stack)

**Note:** It is highly recommended to use try-catch for exception handling instead of throwing exception using throws clause

**Syntax:**

```

try
{
 //code to be monitored for an exception
 throw exception_object
 //throw an exception object to the Java Runtime
}
catch(exception_object)
{
 //exception handler for catching the exception_object
}
finally
{
 //The code will get executed in any case
}

```

**try...catch block:**

**try**

The statements susceptible to a runtime error fall under the try block. A try block can't exist independently. It has to accompany by either catch or finally block.

### catch

The catch block contains the statements of action to be taken when an exception occurs in the try block. Catch used without a try block leads to a compile-time error.

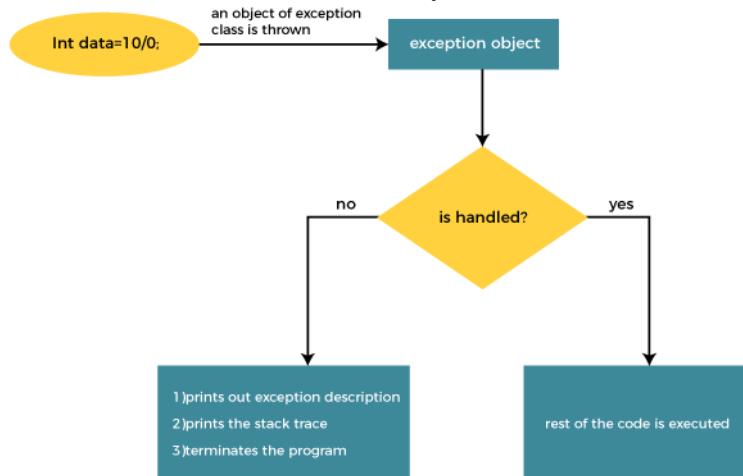


Image: try block

Reference: <https://www.javatpoint.com/try-catch-block>

### Example:

```
class Main_class {
 public static void main(String[] args) {
 int var1 = 32;
 double result;
 try
 {
 result = var1/0;
 System.out.println("Result of division: " +result);
 }
 catch (ArithmeticException e)
 {
 System.out.println("Exception caught! " +e);
 }
 }
}
```

### Output:

```
Exception caught! java.lang.ArithmetiException: / by zero
```

**Explanation:** The moment an exception occurs in the try block, the flow of execution jumps to the matching catch block. For the time, all the statements in between are skipped, and statements under the catch block get executed. Afterward, the control goes back to the try block and continues normal execution.

### Multiple catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

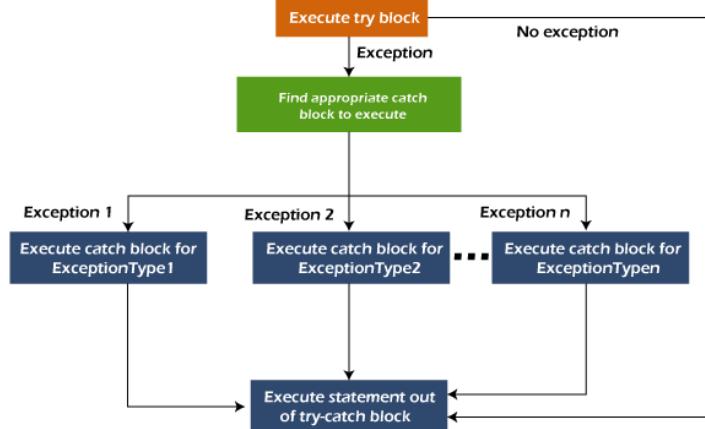


Image: Multiple catch block

Reference: <https://www.javatpoint.com/multiple-catch-block-in-java>

### Example

```

public class MultipleCatchBlock1 {
 public static void main(String[] args) {
 try{
 int a[]={};
 a[5]=30/0;
 }
 catch(ArithmaticException e)
 {
 System.out.println("Arithmatic Exception occurs");
 }
 catch(ArrayIndexOutOfBoundsException e)
 {
 System.out.println("ArrayIndexOutOfBoundsException occurs");
 }
 catch(Exception e)
 {
 System.out.println("Parent Exception occurs");
 }
 System.out.println("rest of the code"); } }

```

### Output

Arithmatic Exception occurs  
rest of the code

### **finally block:**

This block contains a set of statements that executes whether or not an exception is thrown.

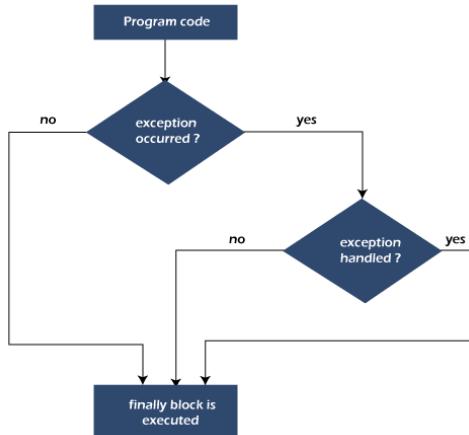


Image: finally block

Reference: <https://www.javatpoint.com/finally-block-in-exception-handling>

### Why use Java finally block?

finally block in Java can be used to put "cleanup" code such as closing a file, closing connection, etc. The important statements to be printed can be placed in the finally block.

### throw and throws Keyword

**throw:** The throw keyword utilizes to throw the exception explicitly.

#### Example:

```

public class ThrowExample{

 void Votingage(int age){
 if(age<18)
 throw new ArithmeticException("you can't vote as not Eligible to
vote");
 else
 System.out.println("Eligible for voting");
 }
 public static void main(String args[]){
 ThrowExample obj = new ThrowExample();
 obj.Votingage(13);
 System.out.println("End Of Program");
 }
}

```

#### Output:

Exception in thread "main" java.lang.ArithmaticException: you can't vote as not Eligible to vote at ThrowExample.Votingage(ThrowExample.java:5) at ThrowExample.main(ThrowExample.java:11)

**throws:** This keyword is used with the method prototype which indicates the type of exceptions that the method might throw to the java runtime.

#### Example:

```

public class ThrowsExample{

```

```

int divion(int a, int b) throws ArithmeticException{
 int intet = a/b;
 return intet;
}

public static void main(String args[]){
 ThrowsExample obj = new ThrowsExample();
 try{
 System.out.println(obj.divion(15,0));
 }
 catch(ArithmeticException e){
 System.out.println("Division cannot be done using ZERO");
 } } }
```

### **Output:**

Division cannot be done using ZERO

### **Multithreading in Java**

**Multithreading in Java** is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc.

### **Advantages of Java Multithreading**

- 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- 2) You **can perform many operations together, so it saves time**.
- 3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

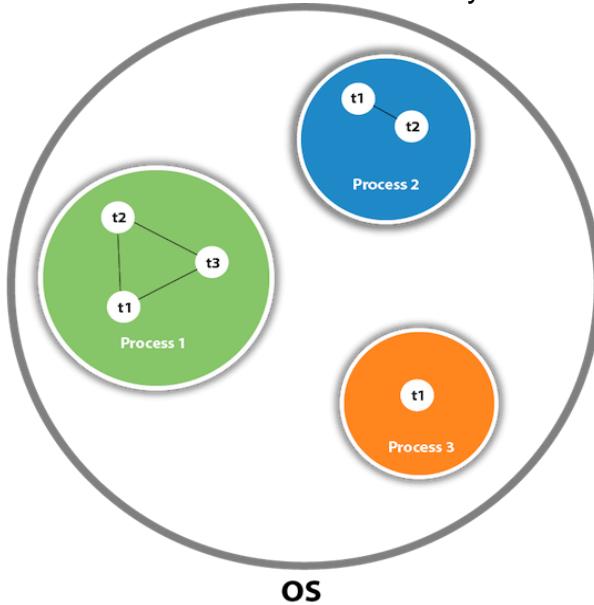
### **Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
  - Thread-based Multitasking (Multithreading)
- 1) Process-based Multitasking (Multiprocessing)
    - Each process has an address in memory. In other words, each process allocates a separate memory area.
    - A process is heavyweight.
    - Cost of communication between the process is high.
    - Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.
  - 2) Thread-based Multitasking (Multithreading)
    - Threads share the same address space.
    - A thread is lightweight.
    - Cost of communication between the thread is low.

## What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



Reference: [Multithreading in Java - javatpoint](#)

As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

## Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

## Java Thread Methods

In Java, the `Thread` class provides several methods to manage and control threads. These methods allow you to perform various operations on threads, such as creating and starting threads, controlling thread execution, pausing and resuming threads, and more. Here are some commonly used methods of the `Thread` class:

### 1. `start()`:

- This method starts the execution of a thread. When called, the `run()` method of the thread is invoked in a separate thread of execution.
- Example: `myThread.start();`

### 2. `run()`:

- This method contains the code that will be executed in a thread. You need to override this method in a subclass of `Thread` or implement the `Runnable` interface and provide the implementation of the `run()` method.

Example:

```
class MyThread extends Thread {
 public void run() {
 // Code to be executed in the thread
 }
}
```

```
}
```

### 3. `sleep()`:

- This method pauses the execution of the current thread for a specified amount of time, allowing other threads to execute.

### 4. `yield()`:

- This method temporarily pauses the execution of the current thread and gives a chance for other threads of the same priority to execute. If no other threads of the same priority are runnable, the thread continues execution.

- Example: `Thread.yield();`

### 5. `join()`:

- This method allows one thread to wait for the completion of another thread. The calling thread will pause until the specified thread has completed its execution.

- Example: `myThread.join();`

### 6. `isAlive()`:

- This method checks if a thread is currently executing (alive) or has terminated.
- Example: `if (myThread.isAlive()) { /\* Thread is still running \*/ }`

### 7. `interrupt()`:

- This method interrupts a thread by setting its interrupt status to true. It can be used to signal a thread to stop its execution gracefully.

- Example: `myThread.interrupt();`

### 8. `isInterrupted()` and `interrupted()`:

- These methods check if a thread has been interrupted. `isInterrupted()` checks the interrupt status without clearing it, while `interrupted()` checks and clears the interrupt status.

Example:

```
if (Thread.currentThread().isInterrupted()) {
 // Thread has been interrupted
}
```

These are just a few examples of the methods provided by the `Thread` class. There are other methods available to manage thread priorities, obtain thread information, synchronize threads, and more. It's important to understand and use these methods appropriately to control and coordinate the execution of threads in your Java programs.

## How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread. JVM calls the run() method on the thread.
3. **public void sleep(long milliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long milliseconds):** waits for a thread to die for the specified milliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(deprecated).
16. **public void resume():** is used to resume the suspended thread(deprecated).
17. **public void stop():** is used to stop the thread(deprecated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

The **start() method** of Thread class is used to start a newly created thread. It performs the following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

## 1) Java Thread Example by extending Thread class

FileName: Multi.java

```

class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi t1=new Multi();
t1.start();
}

}

```

**Output:**

```
thread is running...
```

## 2) Java Thread Example by implementing Runnable interface

**FileName:** Multi3.java

```

class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1); // Using the constructor Thread(Runnable r)
t1.start();
}
}

```

**Output:**

```
thread is running...
```

If you are not extending the Thread class, your class object would not be treated as a thread object. So you need to explicitly create the Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.

## 3) Using the Thread Class: Thread(String Name)

We can directly use the Thread class to spawn new threads using the constructors defined above.

**FileName:** MyThread1.java

```

public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thread(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
}

```

```
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
```

**Output:**

```
My first thread
```

#### 4) Using the Thread Class: Thread(Runnable r, String name)

Observe the following program.

**FileName:** MyThread2.java

```
public class MyThread2 implements Runnable
{
 public void run()
 {
 System.out.println("Now the thread is running ...");
 }

 // main method
 public static void main(String args[])
 {
 // creating an object of the class MyThread2
 Runnable r1 = new MyThread2();

 // creating an object of the class Thread using Thread(Runnable r, String name)
 Thread th1 = new Thread(r1, "My new thread");

 // the start() method moves the thread to the active state
 th1.start();

 // getting the thread name by invoking the getName() method
 String str = th1.getName();
 System.out.println(str);
 }
}
```

**Output:**

```
My new thread
```

```
Now the thread is running ...
```

#### Collections in Java

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

## What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

## What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

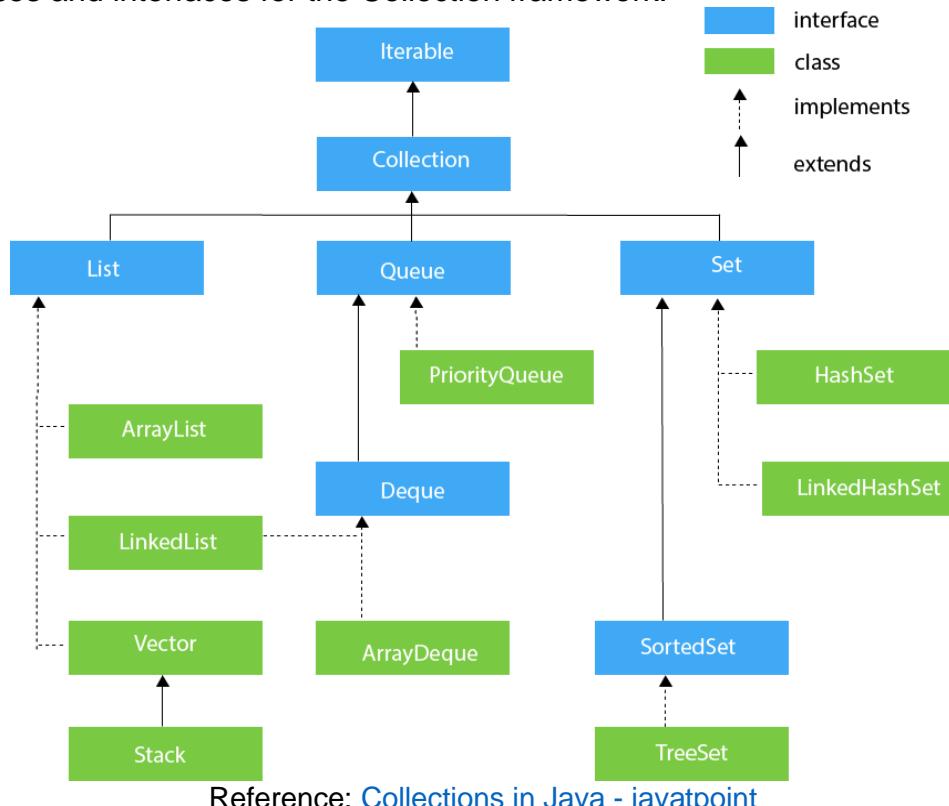
## What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

## Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



### 1. Interfaces:

- **Collection**: This interface defines the general behavior of collections. It provides methods for adding, removing, and accessing elements from a collection.
- **List**: This interface extends the **Collection** interface and represents an ordered collection of elements. It allows duplicate elements and provides methods for accessing elements by index.
- **Set**: This interface extends the **Collection** interface and represents a collection of unique elements. It does not allow duplicate elements.

- **Map:** This interface represents a mapping between keys and values. It associates each key with a corresponding value and does not allow duplicate keys.
- **Queue:** This interface represents a collection that follows the FIFO (First-In-First-Out) principle. It provides methods for adding elements at the end and removing elements from the front.

## 2. Classes:

- **ArrayList:** This class implements the **List** interface using a dynamic array. It provides fast random access but slower insertion and deletion operations.
- **LinkedList:** This class implements the **List** interface using a doubly-linked list. It provides fast insertion and deletion operations but slower random access.
- **HashSet:** This class implements the **Set** interface using a hash table. It provides constant-time performance for basic operations but does not guarantee the order of elements.
- **TreeSet:** This class implements the **Set** interface using a self-balancing binary search tree. It maintains the elements in sorted order.
- **HashMap:** This class implements the **Map** interface using a hash table. It provides constant-time performance for basic operations but does not guarantee the order of elements.
- **TreeMap:** This class implements the **Map** interface using a self-balancing binary search tree. It maintains the key-value pairs in sorted order based on the keys.

## 3. Algorithms and Utilities:

- **Collections:** This class provides various utility methods for working with collections, such as sorting, searching, shuffling, reversing, and more.
- **Arrays:** This class provides utility methods for working with arrays, including methods to convert arrays to collections and vice versa.

## Java Lambda Expressions

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is a concise way to represent a method implementation using functional interfaces. It is a feature introduced in Java 8 that allows you to write more readable and expressive code (or) The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

## Functional Interface

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an annotation `@FunctionalInterface`, which is used to declare an interface as functional interface.

### Why use Lambda Expression

1. To provide the implementation of Functional interface.

## 2. Less coding.

### Java Lambda Expression Syntax *(argument-list) -> {body}*

Java lambda expression is consisted of three components.

- 1) Argument-list: It can be empty or non-empty as well.
- 2) Arrow-token: It is used to link arguments-list and body of expression.
- 3) Body: It contains expressions and statements for lambda expression.

#### No Parameter Syntax

```
() -> {
 //Body of no parameter lambda }
```

#### One Parameter Syntax

```
(p1) -> {
 //Body of single parameter lambda
}
```

#### Two Parameter Syntax

```
(p1,p2) -> {
 //Body of multiple parameter lambda
}
```

Let's see a scenario where we are not implementing Java lambda expression. Here, we are implementing an interface without using lambda expression.

#### Without Lambda Expression Example 1

```
interface MyFunction {
 int apply(int a, int b);
}

public class NoLambdaExample {
 public static void main(String[] args) {
 // Anonymous inner class as implementation of the functional interface
 MyFunction sum = new MyFunction() {
 @Override
 public int apply(int a, int b) {
 return a + b;
 }
 };

 // Using the implementation
 int result = sum.apply(2, 3);
 System.out.println("Sum: " + result);
 }
}
```

#### Java Lambda Expression Example 1

Now, we are going to implement the above example with the help of Java lambda expression.

```
// Functional interface
interface MyFunction {
```

```

 int apply(int a, int b);
 }

public class LambdaExample {
 public static void main(String[] args) {
 // Lambda expression as implementation of the functional interface
 MyFunction sum = (a, b) -> a + b;

 // Using the lambda expression
 int result = sum.apply(2, 3);
 System.out.println("Sum: " + result);
 }
}

```

Output (Same for both Example1)

Sum: 5

### Without Lambda Expression Example 2

```

interface Drawable{
 public void draw();
}

public class LambdaExpressionExample {
 public static void main(String[] args) {
 int width=10;

 //without lambda, Drawable implementation using anonymous class
 Drawable d=new Drawable(){
 public void draw(){System.out.println("Drawing "+width);}
 };
 d.draw();
 }
}

```

### Java Lambda Expression Example 2

Now, we are going to implement the above example with the help of Java lambda expression.

```

@FunctionalInterface //It is optional
interface Drawable{
 public void draw();
}

public class LambdaExpressionExample2 {
 public static void main(String[] args) {
 int width=10;

 //with lambda
 Drawable d2=()->{
 System.out.println("Drawing "+width);
 };
 d2.draw();
 }
}

```

## Output (Same for both Example 2)

Drawing 10

A lambda expression can have zero or any number of arguments. Let's see the examples:

### Java Lambda Expression Example: No Parameter

```
interface Sayable{
 public String say();
}

public class LambdaExpressionExample3{
 public static void main(String[] args) {
 Sayable s=()->{
 return "I have nothing to say.";
 };
 System.out.println(s.say());
 }
}
```

#### Output

I have nothing to say.

### Java Lambda Expression Example: Single Parameter

```
interface Sayable{
 public String say(String name);
}

public class LambdaExpressionExample4{
 public static void main(String[] args) {

 // Lambda expression with single parameter.
 Sayable s1=(name)->{
 return "Hello, "+name;
 };
 System.out.println(s1.say("Alexa"));

 // You can omit function parentheses
 Sayable s2= name ->{
 return "Hello, "+name;
 };
 System.out.println(s2.say("Alexa"));
 }
}
```

#### Output

Hello, Alexa  
Hello, Alexa

### Java Lambda Expression Example: Multiple Parameters

```

interface Addable{
 int add(int a,int b);
}

public class LambdaExpressionExample5{
 public static void main(String[] args) {

 // Multiple parameters in lambda expression
 Addable ad1=(a,b)->(a+b);
 System.out.println(ad1.add(10,20));

 // Multiple parameters with data type in lambda expression
 Addable ad2=(int a,int b)->(a+b);
 System.out.println(ad2.add(100,200));
 }
}

```

### Output

```

30
300

```

## Java Annotations

**Java Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

First, we will learn some built-in annotations then we will move on creating and using custom annotations.

### Built-In Java Annotations

There are several built-in annotations in Java. Some annotations are applied to Java code and some to other annotations.

Built-In Java Annotations used in Java code

- @Override
- @SuppressWarnings
- @Deprecated

### Built-In Java Annotations used in other annotations

- @Target
- @Retention
- @Inherited
- @Documented

### Understanding Built-In Annotations

Let's understand the built-in annotations first.

#### @Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we do the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurance that method is overridden.

```
class Animal{
void eatSomething(){System.out.println("eating something");}
}

class Dog extends Animal{
@Override
void eatsomething(){System.out.println("eating foods");}//should be eatSomething
}

class TestAnnotation1{
public static void main(String args[]){
Animal a=new Dog();
a.eatSomething();
}
}
```

Output:

Output: Compile Time Error

### @SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
import java.util.*;

class TestAnnotation2{
@SuppressWarnings("unchecked")
public static void main(String args[]){

ArrayList list=new ArrayList();
list.add("sonoo");
list.add("vimal");
list.add("ratan");

for(Object obj:list)
System.out.println(obj);

}
}
```

Output:

Now no warning at compile time.

Note:

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

## Java Custom Annotations

Java Custom annotations or Java User-defined annotations are easy to create and use. The `@interface` element is used to declare an annotation. For example:

```
@interface MyAnnotation{}
```

Here, `MyAnnotation` is the custom annotation name.

### Points to remember for java custom annotation signature

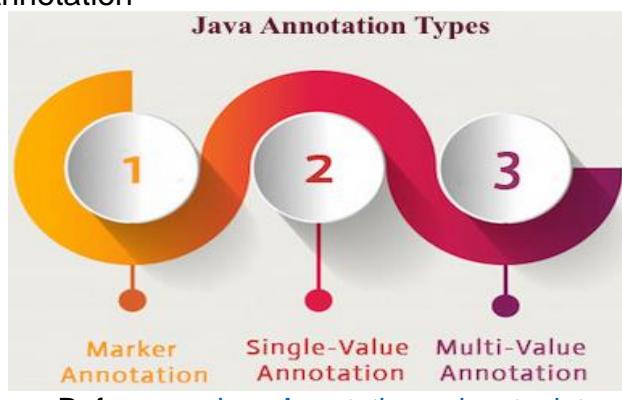
There are few points that should be remembered by the programmer.

1. Method should not have any throws clauses
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach `@` just before interface keyword to define annotation.
5. It may assign a default value to the method.

## Types of Annotation

There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation



Reference: [Java Annotations - javatpoint](#)

### 1) Marker Annotation

An annotation that has no method, is called marker annotation. For example:

```
@interface MyAnnotation{}
```

### 2) Single-Value Annotation

An annotation that has one method, is called single-value annotation. For example:

```
@interface MyAnnotation{
 int value();
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{
 int value() default 0;
}
```

### How to apply Single-Value Annotation

Let's see the code to apply the single value annotation.

```
@MyAnnotation(value=10)
```

The value can be anything.

### 3) Multi-Value Annotation

An annotation that has more than one method, is called Multi-Value annotation. For example:

```
@interface MyAnnotation{
int value1();
String value2();
String value3();
}
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{
int value1() default 1;
String value2() default "";
String value3() default "xyz";
}
```

How to apply Multi-Value Annotation

Let's see the code to apply the multi-value annotation.

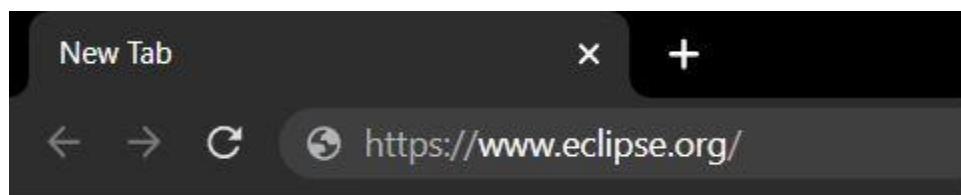
```
@MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")
```

## 2.3 Server Side Scripting Fundamentals

### 2.3.1 Installation of Eclipse IDE on Windows

Follow the below steps to install Eclipse IDE on Windows:

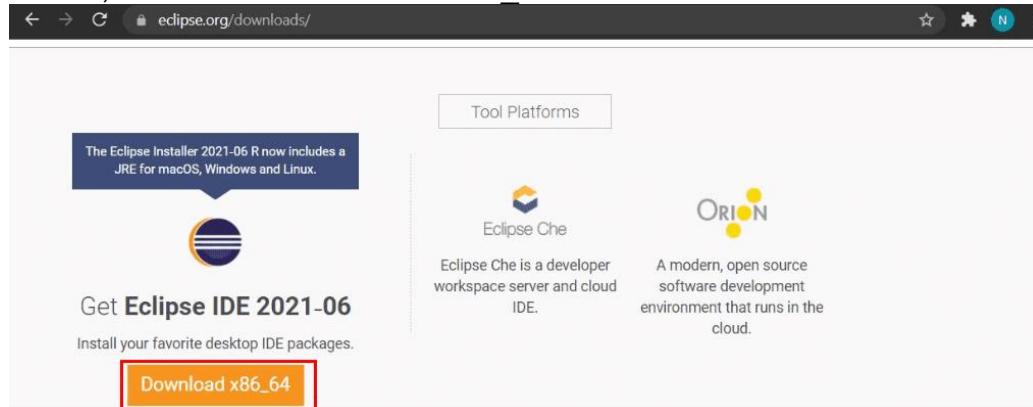
**Step 1:** In the first step, Open your browser and navigate to this <https://www.eclipse.org/>



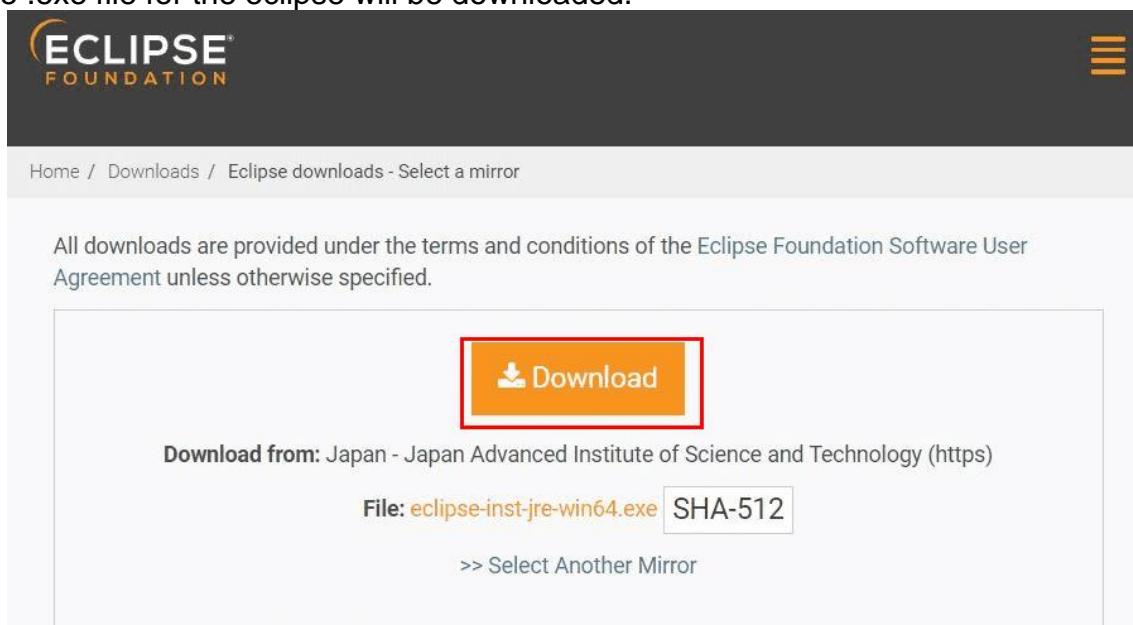
**Step 2:** Then, click on the “Download” button to download Eclipse IDE.



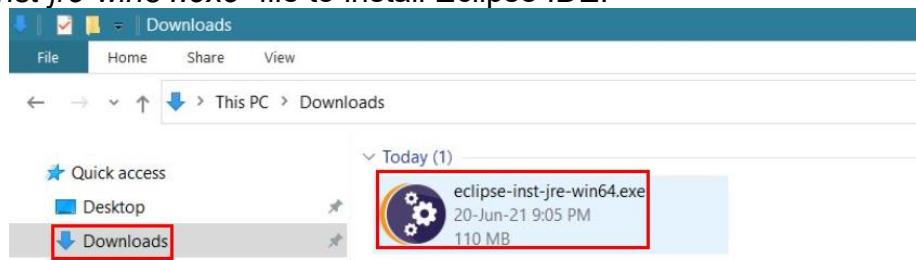
**Step 3:** Now, click on the “Download x86\_64” button.



**Step 4:** Then click on the “Download” button. After clicking on the download button the .exe file for the eclipse will be downloaded.



**Step 5:** Now go to File Explorer and click on “Downloads” after that click on the “eclipse-inst-jre-win64.exe” file to install Eclipse IDE.



**Step 6:** Then, click on “Eclipse IDE for Java Developers”.



**Step 7:** Then, click on the “Install” button.



**Step 8:** Now click on “Create a new Java project”.



Now, you are ready to make new Java projects using eclipse IDE and the screen will look like this :



### 2.3.2 Introduction to Server Side Scripting

Web browsers communicate with web servers using the **HyperText Transfer Protocol (HTTP)**. When you click a link on a web page, submit a form, or run a search, an **HTTP request** is sent from your browser to the target server. The request includes a URL identifying the affected resource, a method that defines the required action (for example to get, delete, or post the resource), and may include additional information encoded in URL parameters (the field-value pairs sent via a query string), as POST data (data sent by the HTTP POST method), or in associated cookies.

Web servers wait for client request messages, process them when they arrive, and reply to the web browser with an **HTTP response** message. The response contains a status line indicating whether or not the request succeeded (e.g. "HTTP/1.1 200 OK" for success). The body of a successful response to a request would contain the requested resource (e.g. a new HTML page, or an image), which could then be displayed by the web browser.

#### Static Site:

The diagram below shows a basic web server architecture for a *static site* (a static site is one that returns the same hard-coded content from the server whenever a particular resource is requested). When a user wants to navigate to a page, the browser sends an HTTP "GET" request specifying its URL.

The server retrieves the requested document from its file system and returns an HTTP response containing the document and a success status (usually 200 OK). If the file cannot be retrieved for some reason, an error status is returned (see client error responses and server error responses).

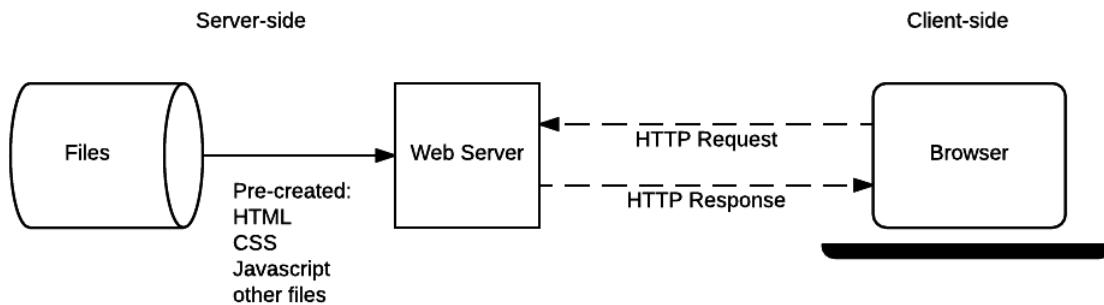


Image: basic static app server

Ref: [Introduction to the server side - Learn web development | MDN \(mozilla.org\)](#)

### Dynamic Site:

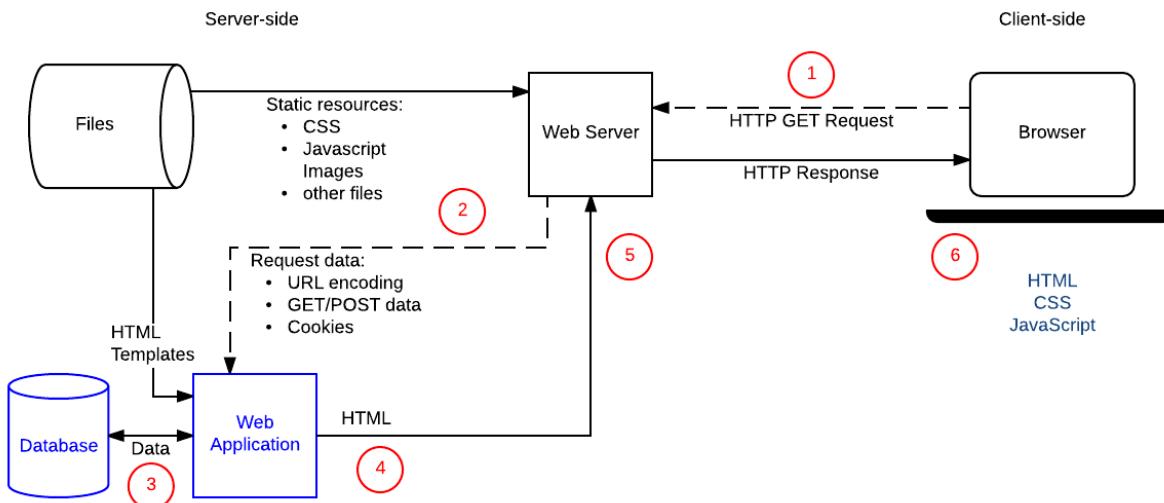
A dynamic website is one where some of the response content is generated *dynamically*, only when needed. On dynamic websites, HTML pages are normally created by inserting data from a database into placeholders in HTML templates (this is a much more efficient way of storing large amounts of content than using static websites).

A dynamic site can return different data for a URL based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g. sending notifications).

Most of the code to support a dynamic website must run on the server. Creating this code is known as "**server-side programming**" (or sometimes "**back-end scripting**").

The diagram below shows a simple architecture for a *dynamic website*. As in the previous diagram, browsers send HTTP requests to the server, then the server processes the requests and returns appropriate HTTP responses.

Requests for *static* resources are handled in the same way as for static sites (static resources are any files that don't change — typically: CSS, JavaScript, Images, pre-created PDF files, etc.).



Reference: [Introduction to the server side - Learn web development | MDN \(mozilla.org\)](https://developer.mozilla.org/en-US/docs/Learn/Server-side)

Requests for dynamic resources are instead forwarded (2) to server-side code (shown in the diagram as a *Web Application*). For "dynamic requests" the server interprets the request, reads required information from the database (3), combines the retrieved data with HTML templates (4), and sends back a response containing the generated HTML (5,6).

### Are Server-side and Client-Side Programming being same?

Let's now turn our attention to the code involved in server-side and client-side programming. In each case, the code is significantly different:

- They have different purposes and concerns.
- They generally don't use the same programming languages (the exception being JavaScript, which can be used on the server- and client-side).
- They run inside different operating system environments.

Code running in the browser is known as **client-side code** and is primarily concerned with improving the appearance and behavior of a rendered web page. This includes selecting and styling UI components, creating layouts, navigation, form validation, etc. By contrast, server-side website programming mostly involves choosing *which content* is returned to the browser in response to requests. The server-side code handles tasks like validating submitted data and requests, using databases to store and retrieve data and sending the correct data to the client as required.

Client-side code is written using HTML, CSS, and JavaScript — it is run inside a web browser and has little or no access to the underlying operating system (including limited access to the file system). Web developers can't control what browser every user might be using to view a website — browsers provide inconsistent levels of compatibility with client-side code features, and part of the challenge of client-side programming is handling differences in browser support gracefully.

Server-side code can be written in any number of programming languages — examples of popular server-side web languages include PHP, Python, Ruby, C#, and JavaScript (NodeJS). The server-side code has full access to the server operating system and the developer can choose what programming language (and specific version) they wish to use.

Developers typically write their code using **web frameworks**. Web frameworks are collections of functions, objects, rules and other code constructs designed to solve common problems, speed up development, and simplify the different types of tasks faced in a particular domain. Again, while both client and server-side code use frameworks, the domains are very different, and hence so are the frameworks. Client-side web frameworks simplify layout and presentation tasks while server-side web frameworks provide a lot of "common" web server functionality that you might otherwise have to implement yourself (e.g. support for sessions, support for users and authentication, easy database access, templating libraries, etc.).

**Note:** Client-side frameworks are often used to help speed up development of client-side code, but you can also choose to write all the code by hand; in fact, writing your code by hand can be quicker and more efficient if you only need a small, simple website UI. In contrast, you would almost never consider writing the server-side component of a web app without a framework — implementing a vital feature like an HTTP server is really hard to do from scratch in say Python, but Python web frameworks like Django provide one out of the box, along with other very useful tools.

### **Advantage of Server side Scripting**

There are several advantages of using server side scripting languages that are as follows:

1. The main advantage of using server side scripting is that all the processing of data takes place before a web page is sent to the browser. As a result, server side script code remains hidden from users.
2. Server side scripting is an independent of browser. That is, browser does not impact on the processing of script code because all the data processing takes place on the server end.
3. It allows database interactivity with web pages.
4. Furthermore, it allows the use of templates for creating HTML web pages. A template is a file that contains HTML code to which contents from a text file, database, and other data retrieves dynamically before displaying the web page to the user.
5. Server side script cannot be disabled at the client side.
6. Server side scripting is more secure than client side scripting.
7. It can access files and database that do not normally be available to the client or user.
8. It provides code reusability.

### **Disadvantage of Server side Scripting**

There are also a few disadvantages of using server side scripts that are as follows:

1. We need a high configuration server to use server side scripting.
2. We require a lot of processing speed for running complex tasks. It possibly slows down the website if the server speed is not good.
3. Server side scripting is slower than client side scripting. It takes too much time because it requires request from the client.

### **Server side Scripting Language Example**

The language in which server side script or program is written using syntax is called server side scripting language or server side programming language. These kinds of scripts are always executed before a web page is loaded on the browser. Some common examples of server-side scripting languages are:

- PHP,

- Python,
- Ruby,
- ASP.NET,
- Perl,
- Go,
- JavaScript
- Java

These scripting languages execute like programming languages at the server end.

### **Application of Server side Scripting**

The server side scripting is used to send the information to the web server to execute the script on the web server. Some common use of server-side scripting are listed, as below:

1. Server side scripting is used to protect the credential (username and password). When the user signs up any login web page first time, username and password are saved in the server. Whenever a user wishes to log in, the web server verifies the credential (username and password) of the user with the username and password stored in the server. If they are matched, the user will be allowed to access the server side resources.
2. Whenever we submit an online application, the data or information entered in the form is stored in the database of the server. Once the information is successfully submitted to the server via online forms, the information from the server can be accessed and represented in the form of report, forms, etc.
3. At server side, we can dynamically edit, add, remove, or update the content of web page at any moment.
4. It is used at backend, where the source code is not visible to the client.
5. Server side script can be used for creating and applying complex business logics.
6. It can be used to customize a web page to make it more useful for individual clients.

### **Why Java:**

Java is a popular choice for server-side scripting, particularly when utilizing JSP, servlets, and JDBC, due to several compelling reasons:

1. Robustness and Reliability: Java is known for its robustness and reliability, making it suitable for building enterprise-grade applications. It offers strong exception handling, automatic memory management, and built-in error checking, ensuring stable and secure server-side scripting.
2. Portability: Java's "write once, run anywhere" principle enables developers to build server-side applications that can run on any platform with a Java Virtual Machine (JVM). This portability eliminates the need to rewrite code for different operating systems, reducing development time and effort.
3. Scalability: Java's inherent scalability enables server-side applications to handle high traffic and concurrent user requests. JSP and servlets, running on a scalable web container such as Apache Tomcat or Jetty, efficiently manage multiple client connections and provide robust support for multi-threading.
4. Separation of Concerns: JSP and servlets facilitate the separation of concerns, where presentation logic (HTML) and business logic (Java code) are clearly distinguished. JSP allows embedding Java code within HTML, making it easier to generate dynamic content. Servlets handle the request

processing and act as controllers, coordinating business logic and data manipulation.

5. Extensive Ecosystem: Java has a vast ecosystem with a wealth of libraries, frameworks, and tools. This ecosystem supports server-side development by providing mature frameworks like Spring MVC, Hibernate, and Struts, which simplify application development, data persistence, and MVC (Model-View-Controller) architecture.
6. Database Connectivity: JDBC is a key factor in Java's suitability for server-side scripting. JDBC provides a standardized API for interacting with databases, allowing developers to seamlessly connect to various relational databases, execute SQL queries, and manipulate data. This enables efficient data retrieval, storage, and management in server-side applications.
7. Enterprise Integration: Java's enterprise focus makes it well-suited for integrating with other enterprise systems and technologies. Java supports various communication protocols and formats, such as HTTP, SOAP, REST, and XML, facilitating integration with external services, databases, messaging systems, and more.
8. Security: Java emphasizes security, offering features like built-in authentication, access control, encryption, and secure socket layers. This makes it a preferred choice for developing secure server-side applications that handle sensitive user data and transactions.

Overall, Java, with its combination of JSP, servlets, and JDBC, provides a powerful and comprehensive platform for server-side scripting. It offers scalability, portability, robustness, and an extensive ecosystem that empowers developers to build robust, secure, and enterprise-grade applications.

### 2.3.3 Moving with Servlets

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page). Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below. There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, HttpServletRequest, HttpServletResponse, etc.

#### Why to Learn Servlet?

- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.
- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.

- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

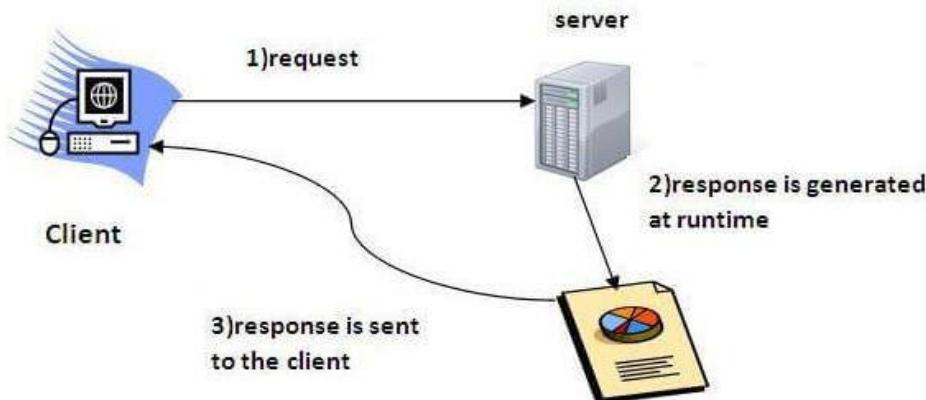
## Applications of Servlet

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

## What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



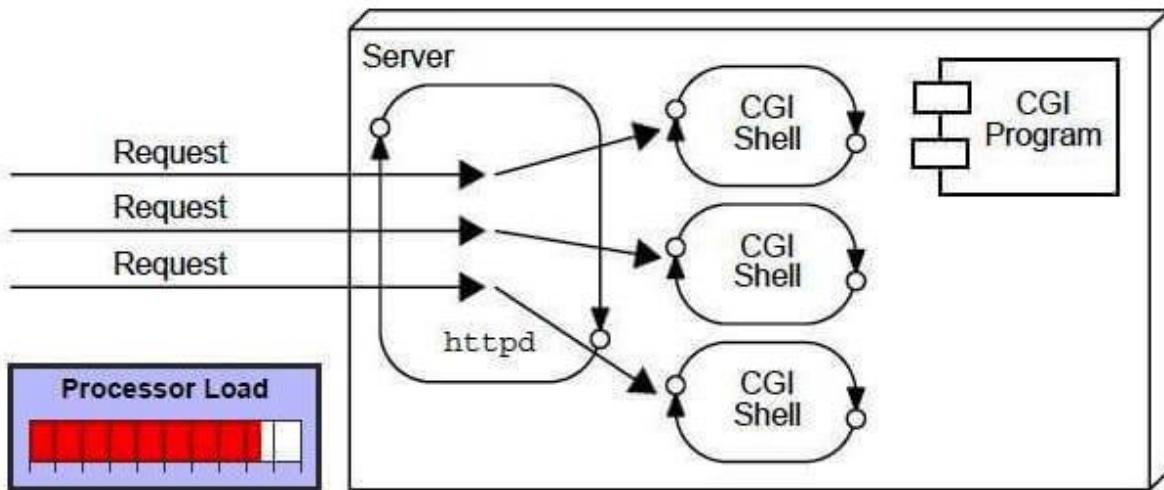
Reference: [Learn Servlet Tutorial - javatpoint](#)

## What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

## CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Reference: [Learn Servlet Tutorial - javatpoint](#)

### Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

### Advantages of Servlet

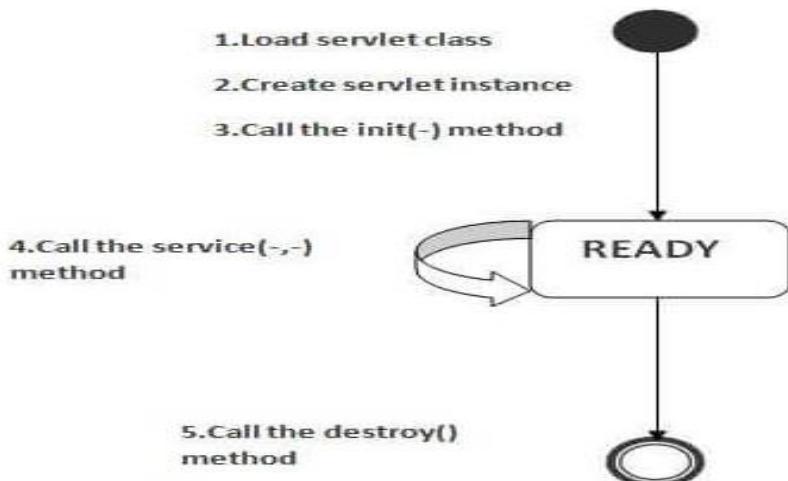
There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

### Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



Reference: [Life cycle of a servlet - javatpoint](#)

The life cycle of a Servlet in Java refers to the various stages that a Servlet goes through during its execution. Servlets are Java classes that extend the `javax.servlet.HttpServlet` class and are used to handle HTTP requests and generate dynamic web content. Here is a detailed explanation of the Servlet life cycle:

#### 1) Servlet class is loaded

The class loader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

#### 2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

#### 3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the `javax.servlet.Servlet` interface. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException
```

#### 4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the `Servlet` interface is given below:

```
public void service(ServletRequest request, ServletResponse response)
 throws ServletException, IOException
```

#### 5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the `Servlet` interface is given below:

```
public void destroy()
```

It is important to note that the Servlet life cycle is managed by the Servlet container, such as Apache Tomcat or Jetty. Developers do not directly control the life cycle methods, except for the `init()` and `destroy()` methods, which can be overridden to provide custom initialization and cleanup logic.

Understanding the Servlet life cycle is crucial for proper Servlet initialization, request handling, and resource management. It allows developers to perform necessary initialization tasks, handle concurrent requests, and release resources appropriately.

### APIs and packages

Servlets in Java are part of the Java Servlet API, which provides classes and interfaces for building web applications. The API is defined by the `javax.servlet` and `javax.servlet.http` packages.

### Servlet API

The `javax.servlet` and `javax.servlet.http` packages represent interfaces and classes for servlet api. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of `javax.servlet` package. Interfaces in `javax.servlet` package

There are many interfaces in `javax.servlet` package. They are as follows:

1. `Servlet`
2. `ServletRequest`
3. `ServletResponse`
4. `RequestDispatcher`
5. `ServletConfig`
6. `ServletContext`
7. `SingleThreadModel`
8. `Filter`
9. `FilterConfig`
10. `FilterChain`
11. `ServletRequestListener`
12. `ServletRequestAttributeListener`
13. `ServletContextListener`
14. `ServletContextAttributeListener`

### Classes in `javax.servlet` package

The `javax.servlet` package in Java provides classes and interfaces for developing servlet-based web applications. Here are some of the important classes in the `javax.servlet` package:

#### 1. `Servlet` (Interface):

- Defines methods for the life cycle management of a servlet, including initialization, request handling, and destruction.
- Methods include `init()`, `service()`, and `destroy()`.

#### 2. `GenericServlet` (Abstract Class):

- An abstract implementation of the `Servlet` interface that provides a generic, protocol-independent implementation.
- Can be extended to create servlets that handle multiple types of requests.
- Subclasses need to override the `service()` method to provide request-specific logic.

### 3. `ServletConfig` (Interface):

- Provides a way to configure initialization parameters for a servlet.
- Allows access to information such as the servlet's name, initialization parameters, and the servlet context.

### 4. `ServletException` (Class):

- An exception that is thrown when a servlet encounters an error during initialization, request handling, or destruction.
- Can be used to handle and report servlet-specific exceptions.

### 5. `ServletInputStream` (Class):

- Represents an input stream for reading request data from a client.
- Provides methods for reading request parameters, headers, and other request-specific information.

### 6. `ServletOutputStream` (Class):

- Represents an output stream for writing response data to a client.
- Allows writing response content, setting headers, and handling other response-specific operations.

### 7. `ServletContext` (Interface):

- Provides a way to interact with the servlet container and access context-specific information.
- Allows access to initialization parameters, MIME type mappings, request dispatchers, and other container-related functionality.

These are some of the key classes in the `javax.servlet` package. They play essential roles in the life cycle management, configuration, and request handling of servlets. By utilizing these classes, you can build dynamic web applications using the Java Servlet API. These are some of the key packages and classes within the Java Servlet API. In addition to these, there are other supporting classes and interfaces that provide functionality for handling filters, listeners, annotations, and more.

## **Interfaces in javax.servlet.http package**

The `javax.servlet.http` package in Java provides interfaces and classes for handling HTTP-specific functionality in servlet-based web applications. Here are some of the important interfaces in the `javax.servlet.http` package:

### 1. `HttpServletRequest` (Interface):

- Extends the `ServletRequest` interface and provides methods for handling HTTP-specific requests.
- Allows retrieving HTTP-specific information such as request parameters, headers, session information, and more.

- Provides methods like `getMethod()`, `getParameter()`, `getHeader()`, and `getSession()`.

## 2. `HttpServletResponse` (Interface):

- Extends the `ServletResponse` interface and provides methods for handling HTTP-specific responses.
- Allows setting HTTP response status codes, headers, and sending response data to the client.
- Provides methods like `setStatus()`, `setHeader()`, `getWriter()`, and `sendRedirect()`.

## 3. ` HttpSession` (Interface):

- Represents a session between the web server and the client.
- Provides methods for managing session attributes, including reading, writing, and removing attributes.
- Allows tracking user-specific data across multiple requests.
- Provides methods like `getAttribute()`, `setAttribute()`, `removeAttribute()`, and `invalidate()`.

## 4. ` HttpSessionListener` (Interface):

- Defines methods to handle session-related events, such as session creation and destruction.
- Can be implemented to perform tasks like tracking active sessions, session attribute management, etc.
- Methods include `sessionCreated()` and `sessionDestroyed()`.

## 5. ` HttpSessionAttributeListener` (Interface):

- Defines methods to handle attribute-related events in a session, such as attribute addition, removal, and modification.
- Can be implemented to perform tasks based on changes to session attributes.
- Methods include `attributeAdded()`, `attributeRemoved()`, and `attributeReplaced()`.

## 6. ` HttpSessionBindingListener` (Interface):

- Defines methods to handle events when an object is bound to or unbound from a session.
- Can be implemented by objects that need to perform actions when they are added or removed from a session.
- Methods include `valueBound()` and `valueUnbound()`.

These interfaces in the `javax.servlet.http` package provide the necessary abstractions and methods to handle HTTP-specific functionality in servlets. They enable you to work with HTTP requests, responses, sessions, and session-related events effectively.

## **HttpServlet class**

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

## Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req, ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

To develop servlet-based applications, you would typically include the Java Servlet API JAR file in your project, which provides the necessary classes and interfaces for building servlets and interacting with the servlet container.

## Steps to create a servlet example

There are given 6 steps to create a **Servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

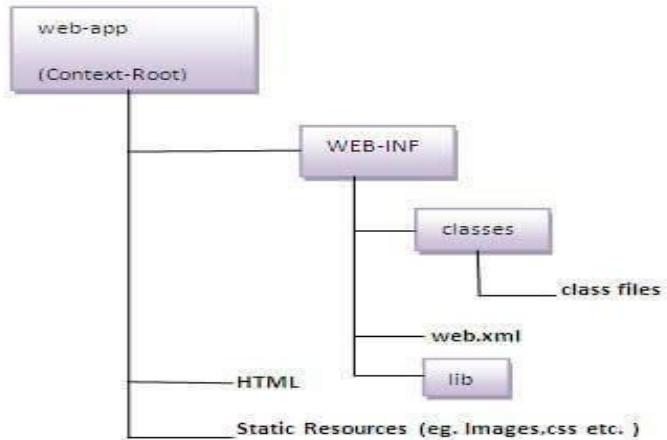
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

### 1)Create a directory structures

The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



Reference: [Servlet Example : Steps to create a servlet example - javatpoint](#)

As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

## 2)Create a Servlet

There are three ways to create the servlet.

- By implementing the Servlet interface
- By inheriting the GenericServlet class
- By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

```
//DemoServlet.java
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class DemoServlet extends HttpServlet{
5. public void doGet(HttpServletRequest req,HttpServletResponse res)
6. throws ServletException,IOException
7. {
8. res.setContentType("text/html");//setting the content type
9. PrintWriter pw=res.getWriter();//get the stream to write the data
10.
11. //writing html in the stream
12. pw.println("<html><body>");
13. pw.println("Servlet Welcome");
14. pw.println("</body></html>");
15.
16. pw.close();//closing the stream
17. }
18.}
```

### 3) Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked. The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

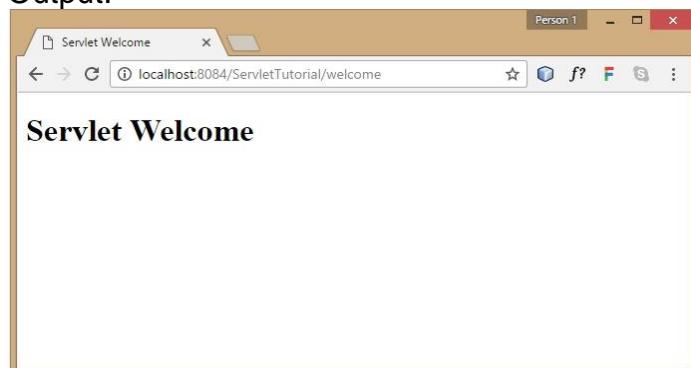
1. <web-app>
2. <servlet>
3. <servlet-name> servletDemo </servlet-name>
4. <servlet-class> DemoServlet </servlet-class>
5. </servlet>
- 6.
7. <servlet-mapping>
8. <servlet-name> servletDemo </servlet-name>
9. <url-pattern> /welcome </url-pattern>
10. </servlet-mapping>
- 11.
12. </web-app>

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

- <web-app> represents the whole application.
- <servlet> is sub element of <web-app> and represents the servlet.
- <servlet-name> is sub element of <servlet> represents the name of the servlet.
- <servlet-class> is sub element of <servlet> represents the class of the servlet.
- <servlet-mapping> is sub element of <web-app>. It is used to map the servlet.
- <url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

Output:



#### 2.3.4 Java Server Pages (JSP)

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

## Why to Learn JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP, etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

## Applications of JSP

As mentioned before, JSP is one of the most widely used language over the web. I'm going to list few of them here:

- JSP vs. Active Server Pages (ASP)  
The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- JSP vs. Pure Servlets  
It is more convenient to write (and to modify!) regular HTML than to have plenty of `println` statements that generate the HTML.

## What is Java Server Pages?

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with `<%` and end with `%>`. A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically. JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

## JSP – Architecture

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development. A JSP container works with the Web server to provide the runtime environment and

other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.

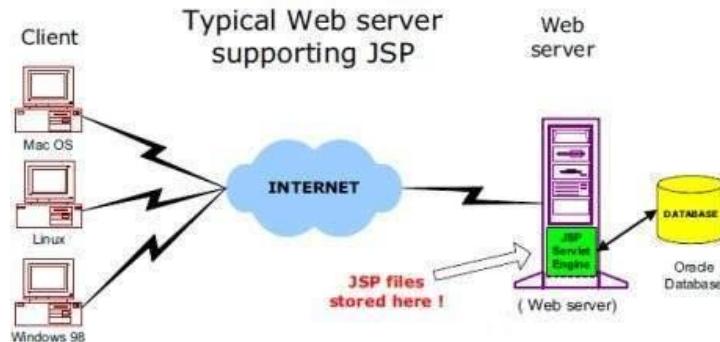


Image: JSP architecture

Reference: [https://www.tutorialspoint.com/jsp/jsp\\_architecture.htm](https://www.tutorialspoint.com/jsp/jsp_architecture.htm)

## JSP Processing

The following steps explain how the web server creates the Webpage using JSP –

1. As with a normal page, your browser sends an HTTP request to the web server.
2. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.
3. The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
5. A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
6. The web server forwards the HTTP response to your browser in terms of static HTML content.
7. Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram –

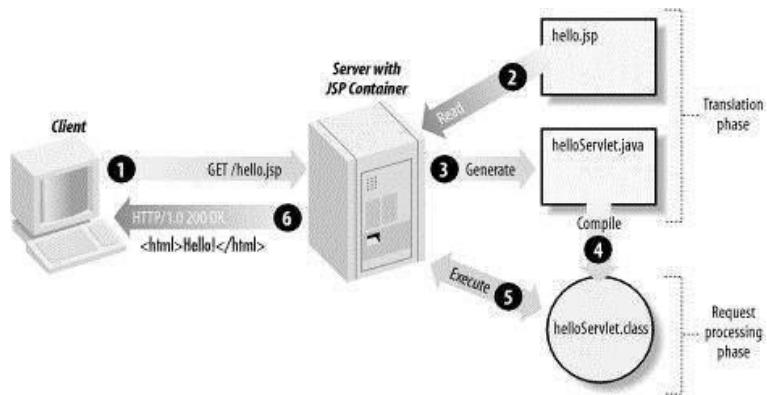


Image: JSP Processing

Reference: [https://www.tutorialspoint.com/jsp/jsp\\_architecture.htm](https://www.tutorialspoint.com/jsp/jsp_architecture.htm)

Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster. So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

## JSP – Lifecycle

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

### Paths Followed By JSP

The following are the paths followed by a JSP –

1. Compilation
2. Initialization
3. Execution
4. Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –

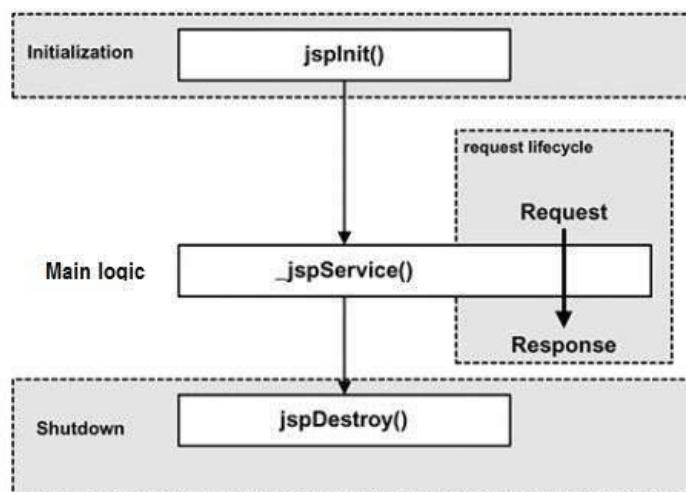


Image: JSP Processing

Reference: [https://www.tutorialspoint.com/jsp/jsp\\_life\\_cycle.htm](https://www.tutorialspoint.com/jsp/jsp_life_cycle.htm)

## JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

1. Parsing the JSP.
2. Turning the JSP into a servlet.
3. Compiling the servlet.

## JSP Initialization

When a container loads a JSP it invokes the `jsplInit()` method before servicing any requests. If you need to perform JSP-specific initialization, override the `jsplInit()` method –

```
public void jsplInit(){
 // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet `init` method, you generally initialize database connections, open files, and create lookup tables in the `jsplInit` method.

## JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the `_jspService()` method in the JSP.

The `_jspService()` method takes an `HttpServletRequest` and an `HttpServletResponse` as its parameters as follows –

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{
 // Service handling code...
}
```

The `_jspService()` method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, GET, POST, DELETE, etc.

## JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The `jspDestroy()` method is the JSP equivalent of the `destroy` method for servlets. Override `jspDestroy` when you need to perform any cleanup, such as releasing database connections or closing open files.

The `jspDestroy()` method has the following form –

```
public void jspDestroy() {
 // Your cleanup code goes here.
}
```

JSP – Syntax

## Elements of JSP

The elements of JSP have been described below –

### The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet –

```
<% code fragment %>
```

You can write the XML equivalent of the above syntax as follows –

```
<jsp:scriptlet>
code fragment
</jsp:scriptlet>
```

Any text, HTML tags, or JSP elements you write must be outside the scriptlet.

Following is the simple and first example for JSP –

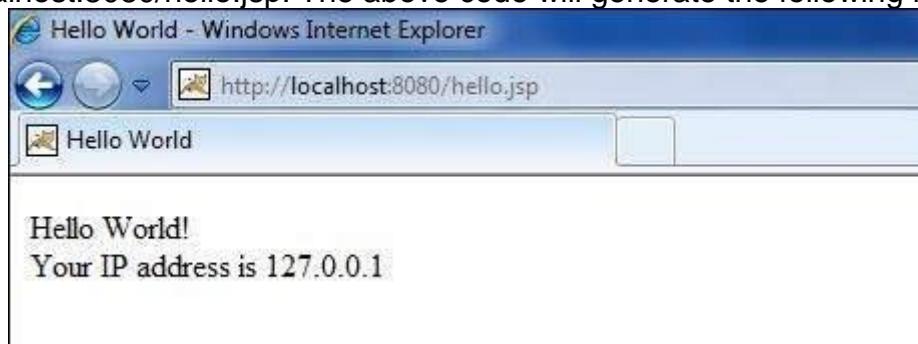
```
<html>
<head><title>Hello World</title></head>

<body>
Hello World!

<%
 out.println("Your IP address is " + request.getRemoteAddr());
%
</body>
</html>
```

NOTE – Assuming that Apache Tomcat is installed in C:\apache-tomcat-7.0.2 and your environment is setup.

Let us keep the above code in JSP file hello.jsp and put this file in C:\apache-tomcat7.0.2\webapps\ROOT directory. Browse through the same using URL <http://localhost:8080/hello.jsp>. The above code will generate the following result –



### JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations –

```
<%! declaration; [declaration;]+ ... %>
```

You can write the XML equivalent of the above syntax as follows –

```
<jsp:declaration>
code fragment
</jsp:declaration>
```

Following is an example for JSP Declarations –

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

### JSP Expression

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression –

```
<%= expression %>
```

You can write the XML equivalent of the above syntax as follows –

```
<jsp:expression>
 expression
</jsp:expression>
```

Following example shows a JSP Expression –

```
<html>
 <head><title>A Comment Test</title></head>

 <body>
 <p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
 </body>
</html>
```

The above code will generate the following result –

```
Today's date: 11-Sep-2010 21:24:25
```

### JSP Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page. Following is the syntax of the JSP comments –

```
<%-- This is JSP comment --%>
```

Following example shows the JSP Comments –

```
<html>
 <head><title>A Comment Test</title></head>

 <body>
 <h2>A Test of Comments</h2>
 <%-- This comment will not be visible in the page source --%>
 </body>
</html>
```

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary –

S.No.	Syntax & Purpose
-------	------------------

---

1

**<%-- comment --%>**

A JSP comment. Ignored by the JSP engine.

---

2

**<!-- comment -->**

An HTML comment. Ignored by the browser.

---

3

**<\%**

Represents static <% literal.

---

4

**%\>**

Represents static %> literal.

---

5

**\'**

A single quote in an attribute that uses single quotes.

---

6

**\"**

A double quote in an attribute that uses double quotes.

## JSP Directives

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

**<%@ directive attribute="value" %>**

There are three types of directive tag –

S.No.	Directive & Description
1	<b>&lt;%@ page ... %&gt;</b> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	<b>&lt;%@ include ... %&gt;</b> Includes a file during the translation phase.
3	<b>&lt;%@ taglib ... %&gt;</b> Declares a tag library, containing custom actions, used in the page

## JSP Actions

JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard

**<jsp:action\_name attribute="value" />**

Action elements are basically predefined functions. Following table lists out the available JSP Actions –

1. **jsp:include** Includes a file at the time the page is requested.
2. **jsp:useBean** Finds or instantiates a JavaBean.
3. **jsp:setProperty** Sets the property of a JavaBean.

- |                           |                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------|
| 4. <b>jsp:getProperty</b> | Inserts the property of a JavaBean into the output.                                    |
| 5. <b>jsp:forward</b>     | Fowards the requester to a new page.                                                   |
| 6. <b>jsp:plugin</b>      | Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin. |
| 7. <b>jsp:element</b>     | Defines XML elements dynamically.                                                      |
| 8. <b>jsp:attribute</b>   | Defines dynamically-defined XML element's attribute.                                   |
| 9. <b>jsp:body</b>        | Defines dynamically-defined XML element's body.                                        |
| 10. <b>jsp:text</b>       | Used to write template text in JSP pages and documents.                                |

### **JSP Implicit Objects**

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are –

<b>S.No.</b>	<b>Object &amp; Description</b>
1	<b>request</b> This is the HttpServletRequest object associated with the request.
2	<b>response</b> This is the HttpServletResponse object associated with the response to the client.
3	<b>out</b> This is the PrintWriter object used to send output to the client.
4	<b>session</b> This is the HttpSession object associated with the request.
5	<b>application</b> This is the ServletContext object associated with the application context.
6	<b>config</b> This is the ServletConfig object associated with the page.
7	<b>pageContext</b> This encapsulates use of server-specific features like higher performance JspWriters.
8	<b>page</b> This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
9	<b>Exception</b> The Exception object allows the exception data to be accessed by designated JSP.

## JSP - Form Processing

When you need to pass some information from your browser to the web server and ultimately to your backend program. The browser uses two methods to pass this information to the web server. These methods are the GET Method and the POST Method.

### The Methods in Form Processing

Let us now discuss the methods in Form Processing.

#### GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows –

***http://www.test.com/hello?key1=value1&key2=value2***

The GET method is the default method to pass information from the browser to the web server and it produces a long string that appears in your browser's Location:box. It is recommended that the GET method is better not used. if you have password or other sensitive information to pass to the server.

The GET method has size limitation: only 1024 characters can be in a request string. This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable which can be handled using getQueryString() and getParameter() methods of request object.

#### POST method

A generally more reliable method of passing information to a backend program is the POST method.

This method packages the information in exactly the same way as the GET method, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

JSP handles this type of requests using getParameter() method to read simple parameters and getInputStream() method to read binary data stream coming from the client.

## Reading Form Data using JSP

JSP handles form data parsing automatically using the following methods depending on the situation –

- `getParameter()` – You call `request.getParameter()` method to get the value of a form parameter.
- `getParameterValues()` – Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- `getParameterNames()` – Call this method if you want a complete list of all parameters in the current request.
- `getInputStream()` – Call this method to read binary data stream coming from the client.

### GET Method Example Using URL

The following URL will pass two values to HelloForm program using the GET method.

***http://localhost:8080/main.jsp?first\_name=ZARA&last\_name=ALI***

Below is the main.jsp JSP program to handle input given by web browser. We are going to use the getParameter() method which makes it very easy to access the passed information –

```
<html>
 <head>
 <title>Using GET Method to Read Form Data</title>
 </head>

 <body>
 <h1>Using GET Method to Read Form Data</h1>

 <p>First Name:
 <%= request.getParameter("first_name")%>
 </p>
 <p>Last Name:
 <%= request.getParameter("last_name")%>
 </p>

 </body>
</html>
```

Now type [http://localhost:8080/main.jsp?first\\_name=ZARA&last\\_name=ALI](http://localhost:8080/main.jsp?first_name=ZARA&last_name=ALI) in your browser's Location:box.

### **GET Method Example Using Form**

Following is an example that passes two values using the HTML FORM and the submit button. We are going to use the same JSP main.jsp to handle this input.

```
<html>
 <body>

 <form action = "main.jsp" method = "GET">
 First Name: <input type = "text" name = "first_name">

 Last Name: <input type = "text" name = "last_name" />
 <input type = "submit" value = "Submit" />
 </form>

 </body>
</html>
```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access <http://localhost:8080>Hello.htm>

### **POST Method Example Using Form**

Let us do a little modification in the above JSP to handle both the GET and the POST method. Below is the main.jsp JSP program to handle the input given by web browser using the GET or the POST methods.

Infact there is no change in the above JSP because the only way of passing parameters is changed and no binary data is being passed to the JSP program. File

handling related concepts will be explained in separate chapter where we need to read the binary data stream.

```
<html>
 <head>
 <title>Using GET and POST Method to Read Form Data</title>
 </head>

 <body>
 <center>
 <h1>Using POST Method to Read Form Data</h1>

 <p>First Name:
 <%= request.getParameter("first_name")%>
 </p>
 <p>Last Name:
 <%= request.getParameter("last_name")%>
 </p>

 </body>
 </html>
```

Following is the content of the Hello.htm file –

```
<html>
 <body>

 <form action = "main.jsp" method = "POST">
 First Name: <input type = "text" name = "first_name">

 Last Name: <input type = "text" name = "last_name" />
 <input type = "submit" value = "Submit" />
 </form>

 </body>
</html>
```

Let us now keep main.jsp and hello.htm in you <Tomcat-access installation directory>/webapps/ROOT directory. When you access <http://localhost:8080>Hello.htm>

### Passing Checkbox Data to JSP Program

Checkboxes are used when more than one option is required to be selected. Following is an example HTML code, CheckBox.htm, for a form with two checkboxes.

```
<html>
 <body>

 <form action = "main.jsp" method = "POST" target = "_blank">
 <input type = "checkbox" name = "maths" checked = "checked" /> Maths
```

```

<input type = "checkbox" name = "physics" /> Physics
<input type = "checkbox" name = "chemistry" checked = "checked" />
Chemistry
<input type = "submit" value = "Select Subject" />
</form>
</body>
</html>

```

Following is main.jsp JSP program to handle the input given by the web browser for the checkbox button.

```

<html>
 <head>
 <title>Reading Checkbox Data</title>
 </head>

 <body>
 <h1>Reading Checkbox Data</h1>

 <p>Maths Flag:
 <%= request.getParameter("maths")%>
 </p>
 <p>Physics Flag:
 <%= request.getParameter("physics")%>
 </p>
 <p>Chemistry Flag:
 <%= request.getParameter("chemistry")%>
 </p>

 </body>
</html>

```

## Reading All Form Parameters

Following is a generic example which uses `getParameterNames()` method of `HttpServletRequest` to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using the `hasMoreElements()` method to determine when to stop and using the `nextElement()` method to get each parameter name.

```

<%@ page import = "java.io.* ,java.util.*" %>

<html>
 <head>
 <title>HTTP Header Request Example</title>
 </head>

 <body>
 <center>

```

```

<h2>HTTP Header Request Example</h2>
<table width = "100%" border = "1" align = "center">
 <tr bgcolor = "#949494">
 <th>Param Name</th>
 <th>Param Value(s)</th>
 </tr>
 <%
 Enumeration paramNames = request.getParameterNames();
 while(paramNames.hasMoreElements()) {
 String paramName = (String)paramNames.nextElement();
 out.print("<tr><td>" + paramName + "</td>\n");
 String paramValue = request.getHeader(paramName);
 out.println("<td> " + paramValue + "</td></tr>\n");
 }
 %>
</table>
</center>

</body>
</html>

```

Following is the content of the Hello.htm –

```

<html>
 <body>

 <form action = "main.jsp" method = "POST" target = "_blank">
 <input type = "checkbox" name = "maths" checked = "checked" /> Maths
 <input type = "checkbox" name = "physics" /> Physics
 <input type = "checkbox" name = "chemistry" checked = "checked" /> Chem
 <input type = "submit" value = "Select Subject" />
 </form>

 </body>
</html>

```

Now try calling JSP using the above Hello.htm

### Expression Language (EL) in JSP

The Expression Language (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc. There are many implicit objects, operators and reserve words in EL. It is the newly added feature in JSP technology version 2.0.

Syntax for Expression Language (EL)

`${ expression }`

EL expressions are a powerful tool that can be used to simplify and improve the readability of JSP code. They can also be used to access data from a variety of sources, making them a versatile and powerful tool for JSP developers.

**Here are some of the benefits of using EL in JSP:**

- EL expressions are more concise than Java code, making JSP pages easier to read and maintain.

- EL expressions are easier to understand for non-technical users, making JSP pages more accessible.
- EL expressions can be used to access data from a variety of sources, making them more versatile than Java code.

If you are new to JSP, I recommend learning about EL expressions early on. They are a powerful tool that can help you to write more concise, readable, and accessible JSP pages.

## Implicit Objects in Expression Language (EL)

There are many implicit objects in the Expression Language. They are as follows:

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope
requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

### EL param example

In this example, we have created two files index.jsp and process.jsp. The index.jsp file gets input from the user and sends the request to the process.jsp which in turn prints the name of the user using EL.

index.jsp

```
<form action="process.jsp">
Enter Name:<input type="text" name="name" />

<input type="submit" value="go"/>
</form>
```

process.jsp

```
Welcome, ${ param.name }
```

## JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

### Advantage of JSTL

- Fast Development** JSTL provides many tags that simplify the JSP.
- Code Reusability** We can use the JSTL tags on various pages.
- No need to use scriptlet tag** It avoids the use of scriptlet tag.

## JSTL Tags

There JSTL mainly provides five types of tags:

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is <a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a> . The prefix of core tag is <b>c</b> .
Function tags	The functions tags provide support for string manipulation and string length. The URL for the functions tags is <a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a> and prefix is <b>fn</b> .
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is <a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a> and prefix is <b>fmt</b> .
XML tags	The XML tags provide flow control, transformation, etc. The URL for the XML tags is <a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a> and prefix is <b>x</b> .
SQL tags	The JSTL SQL tags provide SQL support. The URL for the SQL tags is <a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a> and prefix is <b>sql</b> .

JSTL tags can be used to simplify and improve the readability of JSP code. They can also be used to access data from a variety of sources, making them a versatile and powerful tool for JSP developers.

Here are some of the benefits of using JSTL in JSP:

- JSTL tags are more concise than Java code, making JSP pages easier to read and maintain.
- JSTL tags are easier to understand for non-technical users, making JSP pages more accessible.
- JSTL tags can be used to access data from a variety of sources, making them more versatile than Java code.

## Handling Cookies in JSP

### What are Cookies?

- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlet technology
- The cookies are set in the HTTP Header.
- If the browser is configured to store cookies, it will keep information until expiry date.

### JSP cookies methods

Following are the cookies methods:

- **Public void setDomain(String domain)**

This JSP set cookie is used to set the domain to which the cookie applies

- **Public String getDomain()**

This JSP get cookie is used to get the domain to which cookie applies

- **Public void setMaxAge(int expiry)**

It sets the maximum time which should apply till the cookie expires

- **Public intgetMaxAge()**

It returns the maximum age of cookie in JSP

- **Public String getName()**

It returns the name of the cookie

- **Public void setValue(String value)**

Sets the value associated with the cookie

- **Public String getValue()**

Get the value associated with the cookie

- **Public void setPath(String path)**

This set cookie in JSP sets the path to which cookie applies

- **Public String getPath()**

It gets the path to which the cookie applies

- **Public void setSecure(Boolean flag)**

It should be sent over encrypted connections or not.

- **Public void setComment(String cmt)**

It describes the cookie purpose

- **Public String getComment()**

It the returns the cookie comments which has been described.

## How to Handle Cookies in JSP

1. Creating the cookie object
2. Setting the maximum age
3. Sending the cookie in HTTP response headers

Example:

In this JSP cookies example, we will learn how to call cookie constructor in JSP by creating cookies of username and email, and add age to the cookie for 10 hours and trying to get the variable names in the action\_cookie.jsp

Action\_cookie.jsp.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Cookie</title>
</head>
<body>
<form action="action_cookie_main.jsp" method="GET">
Username: <input type="text" name="username">

Email: <input type="text" name="email" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

### Action\_cookie\_main.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%
Cookie username = new Cookie("username",
 request.getParameter("username"));
Cookie email = new Cookie("email",
 request.getParameter("email"));

username.setMaxAge(60*60*10);
email.setMaxAge(60*60*10);

// Add both the cookies in the response header.
response.addCookie(username);
response.addCookie(email);
%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Cookie JSP</title>
</head>
<body>

Username:
<%= request.getParameter("username")%>
Email:
<%= request.getParameter("email")%>

</body>
</html>
```

### Explanation of the code:

#### Action\_cookie.jsp

Code Line 10-15: Here we are taking a form which has to be processed in action\_cookie\_main.jsp. Also, we are taking two fields “username” and “email” which has to be taken input from the user with a submit button.

#### Action\_cookie\_main.jsp

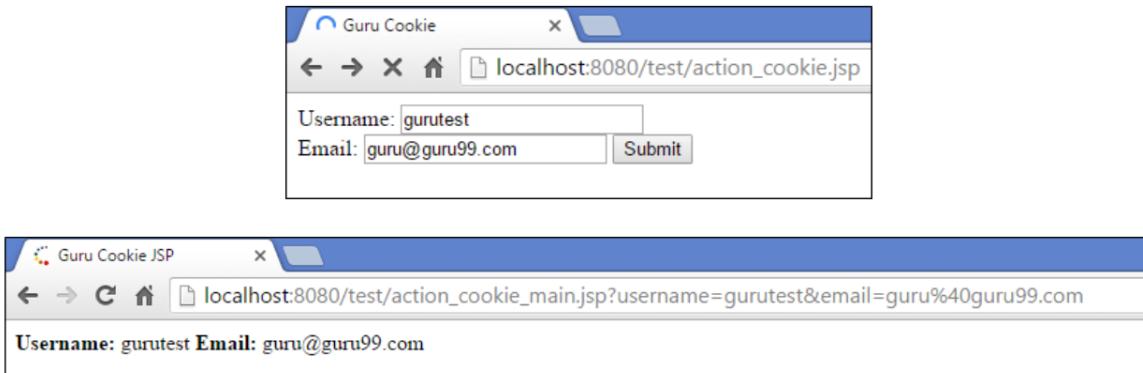
**Code Line 6-9:** Creating two cookie objects of “username” and “email” using request.getParameter.

**Code Line 12-13:** Here we are adding age to both the cookies, which have been created of 10 hours i.e. cookies will expire in that age.

**Code Line 16-17:** Adding cookies to the session of username and email and these two cookies can be fetched when requested by getParameter().

## **Output:**

When you execute the above code you get the following output:



When we execute the action\_cookie.jsp we get two fields username and email, and it takes user input and then we click on the submit button.

We get the output from action\_cookie\_main.jsp where variables are stored in the cookies JSP on the client side.

## **Session Tracking in Java**

In the world of the web, a session is the amount of time in which any two systems interact with each other. Those two systems can have a peer-to-peer or client-server relationship with each other. However, the problem is, in HTTP protocol, the state of the communication is not maintained, i.e., HTTP is a stateless protocol. Session Tracking in Java is used to tackle this problem with the help of servlets.

## **Implementation of Session Tracking in Java**

The following programs shows how to implement session tracking. In the example, we have created four files.

- 1) index.html
- 2) web.xml
- 3) HTTPServletEx1.java
- 4) HTTPServletEx2.java

### **File Name: index.html**

```
1. <html>
2. <head>
3. <body>
4. <form action = "servletA">
5. UserName: <input type = "text" name = "userName"/>

6. <input type = "submit" value = "Press the Button"/>
7. </form>
8. </body>
9. </html>
10.</textarea></div>
11.<p>FileName: web.xml</p>
12.<div class="codeblock"><textarea name="code" class="xml">
13.<web-app >
14.<servlet >
```

```
15.<servlet-name>a1</servlet-name>
16.<servlet-class>HTTPServletEx1</servlet-class>
17.</servlet>
18.<servlet-mapping>
19.<servlet-name>a1</servlet-name>
20.<url-pattern>/servletA</url-pattern>
21.</servlet-mapping>
22.<servlet>
23.<servlet-name>a2</servlet-name>
24.<servlet-class>HTTPServletEx2</servlet-class>
25.</servlet>
26.<servlet-mapping>
27.<servlet-name>a2</servlet-name>
28.<url-pattern>/servletB</url-pattern>
29.</servlet-mapping>
30.</web-app>
```

**File Name:** web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
 version="3.1">

 <servlet>
 <servlet-name>a1</servlet-name>
 <servlet-class>HTTPServletEx1</servlet-class>
 </servlet>

 <servlet-mapping>
 <servlet-name>a1</servlet-name>
 <url-pattern>/servletA</url-pattern>
 </servlet-mapping>

 <servlet>
 <servlet-name>a2</servlet-name>
 <servlet-class>HTTPServletEx2</servlet-class>
 </servlet>

 <servlet-mapping>
 <servlet-name>a2</servlet-name>
 <url-pattern>/servletB</url-pattern>
 </servlet-mapping>

</web-app>
```

**File Name:** HTTPServletEx1.java

```

1. // import statements
2. import java.io.*;
3. import jakarta.servlet.*;
4. import jakarta.servlet.http.*;
5. public class HTTPServletEx1 extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response)
 e)
7. {
8. try {
9. // content type has been to text
10. String contentType = "text/html";
11. response.setContentType(contentType);
12. PrintWriter o = response.getWriter();
13. // the name variable stores the content of the field userName
14. // mentione in the form
15. String name = request.getParameter("userName");
16. // displaying the username
17. o.println("Welcome to JavaTpoint " + name + "!");
18. // a new session is created
19. HttpSession httpSession = request.getSession();
20. // the variable uname contains the value present
21. // in the variable name. The variable uname is
22. // shared to the other servlets present in the application
23. httpSession.setAttribute("uname", name);
24. // Link to reach the other servlet
25. o.print(" Press Here ");
26. o.close();
27. }
28. catch (Exception e)
29. {
30. System.out.println(e);
31. }
32.}
33.

```

### **File Name: HTTPServletEx2.java**

```

1. // import statements
2. import jakarta.servlet.*;
3. import jakarta.servlet.http.*;
4. import java.io.*;
5. public class HTTPServletEx2 extends HttpServlet
6. {
7. public void doGet(HttpServletRequest request, HttpServletResponse response)
 e)
8. {
9. try {
10. {
11. // content type has been to text
12. String contentType = "text/html";

```

```

13. response.setContentType(contentType);
14. PrintWriter o = response.getWriter();
15. HttpSession sessn = request.getSession(false);
16./*We have resumed the session
17.that was created in the previous servlet with the help of the getSession metho
d
18.Note that the parameter that is being passed is 'false'
19.which is to ensure that a new session is not getting created, as
20.we have already got an existing session.
21.*/
22.// getting the name from uname that was created in the
23.// previous servlet
24.String name = (String)sessn.getAttribute("uname");
25.// printing the name and message
26.o.print(name + " you have reached the second page.");
27.o.close();
28.}
29.catch (Exception e)
30.{
31. System.out.println(e);
32.}
33.}
34.}

```

### Follow the steps given below to run the program.

**Step 1:** Install the Apache Tomcat application. Go inside the webapps folder of the Tomcat application, and create a folder of your choice. We have created a MyProject folder.

This PC > Local Disk (C:) > Program Files > Apache Software Foundation > Tomcat 10.0 > webapps			
Name	Date modified	Type	Size
docs	6/16/2021 4:59 AM	File folder	
examples	6/16/2021 4:59 AM	File folder	
host-manager	6/16/2021 4:59 AM	File folder	
manager	6/16/2021 4:59 AM	File folder	
MyProject	6/17/2021 8:24 PM	File folder	
ROOT	6/16/2021 4:59 AM	File folder	

**Step 2:** Inside the MyProject folder, create a WEB-INF folder, and inside the WEB-INF folder, create a classes folder.

**Step 3:** Now, Compile the above-mentioned Java files using the javac command. Keep the generated .class files in the classes folder.

This PC > Local Disk (C:) > Program Files > Apache Software Foundation > Tomcat 10.0 > webapps > MyProject > WEB-INF > classes			
Name	Date modified	Type	Size
HTTPServletEx1.class	6/16/2021 11:47 PM	CLASS File	2 KB
HTTPServletEx1.java	6/16/2021 11:46 PM	JAVA File	2 KB
HTTPServletEx2.class	6/16/2021 11:47 PM	CLASS File	2 KB
HTTPServletEx2.java	6/16/2021 11:46 PM	JAVA File	2 KB

**Step 4:** Now, move outside the classes folder, and create the web.xml file in the WEB-INF folder. Observe the following snapshot.

File Explorer showing the contents of the WEB-INF folder:

Name	Date modified	Type	Size
classes	6/17/2021 8:24 PM	File folder	
web.xml	6/16/2021 11:59 PM	XML Document	1 KB

### Step 5: Along with the WEB-INF folder, keep the index.html file.

File Explorer showing the contents of the MyProject folder:

Name	Date modified	Type	Size
WEB-INF	6/17/2021 8:24 PM	File folder	
index.html	6/16/2021 10:57 PM	Chrome HTML Do...	1 KB

### Step 6: Our application setup is ready. Now, we have to launch the application. To do so, move to the bin folder.

File Explorer showing the contents of the Tomcat 10.0 folder:

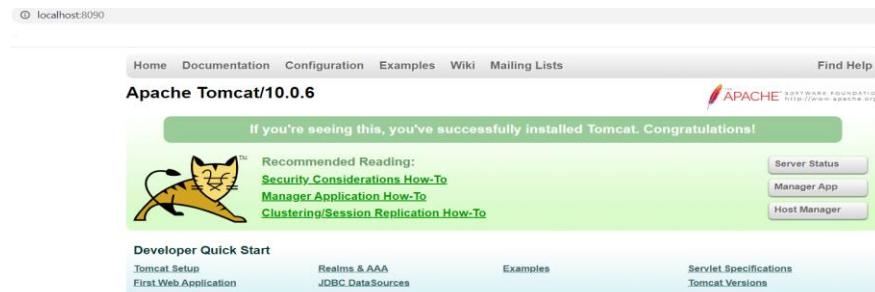
Name	Date modified	Type	Size
bin	6/16/2021 4:59 AM	File folder	
conf	6/16/2021 4:59 AM	File folder	
lib	6/16/2021 8:41 PM	File folder	
logs	6/17/2021 8:27 PM	File folder	
temp	6/16/2021 4:59 AM	File folder	
webapps	6/17/2021 8:50 PM	File folder	
work	6/16/2021 4:59 AM	File folder	
LICENSE	5/8/2021 8:54 PM	File	60 KB
NOTICE	5/8/2021 8:54 PM	File	3 KB
RELEASE-NOTES	5/8/2021 8:54 PM	File	7 KB
tomcat.ico	5/8/2021 8:54 PM	Icon	22 KB
Uninstall.exe	5/8/2021 8:56 PM	Application	82 KB

### Step 7: Inside the bin folder, click on the Tomcat10.exe

File Explorer showing the contents of the bin folder:

Name	Date modified	Type	Size
bootstrap.jar	5/8/2021 8:54 PM	Executable Jar File	34 KB
catalina.bat	5/8/2021 8:54 PM	Windows Batch File	16 KB
ciphers.bat	5/8/2021 8:54 PM	Windows Batch File	3 KB
configtest.bat	5/8/2021 8:54 PM	Windows Batch File	2 KB
digest.bat	5/8/2021 8:54 PM	Windows Batch File	3 KB
makebase.bat	5/8/2021 8:54 PM	Windows Batch File	4 KB
migrate.bat	5/8/2021 8:54 PM	Windows Batch File	3 KB
service.bat	5/8/2021 8:54 PM	Windows Batch File	9 KB
setclasspath.bat	5/8/2021 8:54 PM	Windows Batch File	4 KB
shutdown.bat	5/8/2021 8:54 PM	Windows Batch File	2 KB
startup.bat	5/8/2021 8:54 PM	Windows Batch File	2 KB
Tomcat10.exe	5/8/2021 8:54 PM	Application	128 KB
Tomcat10w.exe	5/8/2021 8:54 PM	Application	118 KB
tomcat-juli.jar	5/8/2021 8:54 PM	Executable Jar File	46 KB
tool-wrapper.bat	5/8/2021 8:54 PM	Windows Batch File	5 KB
version.bat	5/8/2021 8:54 PM	Windows Batch File	2 KB

### Step 8: The Apache Tomcat server has been launched. The Tomcat server usually listens on port number 8090. Therefore, we have to provide the same port number in the URL. To do so, go to the browser and in the URL bar, write localhost:8090, and press enter. You will see the following.



**Step 9:** Now, add `/MyProject` to the URL. Thus, the new URL will be `localhost:8090/MyProject`. After pressing the enter button, the `index.html` file comes into action, and the form is shown on the browser.

← → ⏪ ⓘ localhost:8090/MyProject/

UserName:

**Step 10:** Now write the name of your choice, and click on "Press the Button", we get the following.

← → ⏪ ⓘ localhost:8090/MyProject/servletA?userName=Nikhil+Kumar+

Welcome to JavaTpoint Nikhil Kumar ! [Press Here](#)

**Step 11:** Observe the URL, it shows `servletA`. It is because of the `action` attribute present in the `index.html` file. Now click on "Press Here".

← → ⏪ ⓘ localhost:8090/MyProject/servletB

Nikhil Kumar you have reached the second page.

**Explanation:** In the above code, the `getAttribute()` and `setAttribute()` methods are from the `HttpSession` interface. The `setAttribute()` method creates an attribute in the session scope of the first servlet, and the `getAttribute()` receives the same attribute in the session scope of the second servlet. That's why the same is reflected on the `servletA` as well as on the `servletB`.

### Advantages of Using Http Sessions in a Servlet

- 1) Various sorts of objects can be kept in the session, such as dataset, database, and text.
- 2) Unlike cookies, the dependency of the client's browser has been completely removed on the usage of the sessions. To achieve that, a session object is kept on the server instead of on the client's machine.
- 3) Sessions are transparent and secure.

# Chapter 3: Handling Data using Data Base Management System

## Learning Outcomes:

- Understand the fundamental concepts of Database Management System
- Gain knowledge of DDL, DML, DQL
- Develop Database to store important data for any application
- Create connection between server side and database

### 3.1 Introduction to Database Management System

#### What is a database?

A database is a collection of related information. Modern databases contain millions or even trillions of pieces of information. Databases provide convenience for easy storage and access to data. The word 'datum' means a single piece of information. The word data is the plural form of datum. One of the most important aspects of a database is to easily manage and operate large amounts of data.

#### What is a Database Management System?

A Relational Database Management System is a tabular based collection of programs and capabilities that provides an interface between users and applications and the database, offering a systematic way to create, update, delete, manage, and retrieve data. Most relational database management systems use the SQL programming language to access the database and many follow the ACID (Atomicity, Consistency, Isolation, Durability) properties of the database:

- Atomicity: If any statement in the transaction fails, the entire transaction fails and the database is left unchanged.
- Consistency: The transaction must meet all protocols defined by the system -- no partially completed transactions.
- Isolation: No transaction has access to any other transaction that is unfinished. Each transaction is independent.
- Durability: Once a transaction has been committed, it will remain committed through the use of transaction logs and backups.

## What is a relational database?

A database that follows the relational model and stores data in a tabular format is known as a relational database. The database has rows and columns and a unique key for each data point.

Relational databases are very common and have high usage. Pretty much everything you have entered online in a form or something like that usually gets stored in a relational database.

**Examples of relational databases:** Microsoft SQL Server, Oracle, MYSQL.

A relational database is one that stores data in tables. The relationship between each data point is clear and searching through those relationships is relatively easy. The relationship between tables and field types is called a schema. For relational databases, the schema must be clearly defined. Let's look at an example:

Name	Dry/Wet Food	Good Boy (Y/N)
Fido	Dry	Y
Rex	Wet	N
Bubbles	Dry	Y
Cujo	Wet	N

Tag #	Height (in)	Weight (lbs)
1573	15	21
2684	9	7
3795	27	130
4806	6	5

Tag #	Name	Breed	Color	Age
1573	Fido	Beagle	Brown/White	1.5
2684	Rex	Pekingese	White	9
3795	Bubbles	Rottweiler	Black	5
4806	Cujo	Chihuahua	Gold	4

Image: Relational Database

Reference: <https://insightsoftware.com/blog/whats-the-difference-relational-vs-non-relational-databases/>

## Advantages of a Relational Database

The main advantage of a relational database is its formally described, tabular structure, from which data can be easily stored, categorized, queried, and filtered without needing to reorganize database tables. Further benefits of relational databases include:

- Scalability: New data may be added independent of existing records.
- Simplicity: Complex queries are easy for users to perform with SQL.
- Data Accuracy: Normalization procedures eliminate design anomalies.
- Data Integrity: Strong data typing and validity checks ensure accuracy and consistency.
- Security: Data in tables within a RDBMS can limit access to specific users.
- Collaboration: Multiple users can access the same database concurrently.

## Why uses a database?

- Databases can store very large numbers of records efficiently (they take up little space).
- It is very quick and easy to find information.
- It is easy to add new data and to edit or delete old data.

- Data can be searched easily, eg 'find all Ford cars'.
- Data can be sorted easily, for example into 'date first registered' order.
- Data can be imported into other applications, for example a mail-merge letter to a customer saying that an MOT test is due.
- More than one person can access the same database at the same time - multi-access.
- Security may be better than in paper files

## Why do we need a Database?

A database is a collection of data, usually stored in electronic form. A database is typically designed so that it is easy to store and access information. A good database is crucial to any company or organisation. This is because the database stores all the pertinent details about the company such as employee records, transactional records, salary details etc.

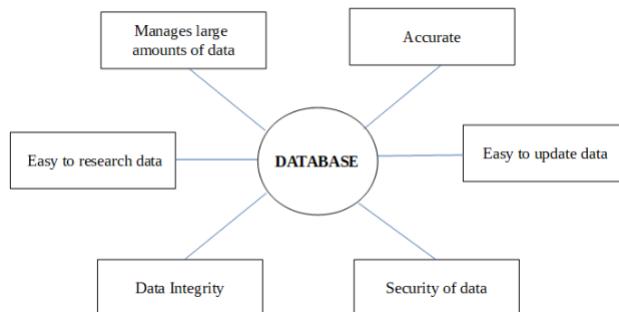


Image: Why do we need a Database

Reference: <https://www.tutorialspoint.com/Why-do-we-need-a-Database>

The various reasons a database is important are –

- **Manages large amounts of data**  
A database stores and manages a large amount of data on a daily basis. This would not be possible using any other tool such as a spreadsheet as they would simply not work.
- **Accurate**  
A database is pretty accurate as it has all sorts of build in constraints, checks etc. This means that the information available in a database is guaranteed to be correct in most cases.
- **Easy to update data**  
In a database, it is easy to update data using various Data Manipulation languages (DML) available. One of these languages is SQL.
- **Security of data**  
Databases have various methods to ensure security of data. There are user logins required before accessing a database and various access specifiers. These allow only authorised users to access the database.
- **Data integrity**  
This is ensured in databases by using various constraints for data. Data integrity in databases makes sure that the data is accurate and consistent in a database.
- **Easy to research data**

It is very easy to access and research data in a database. This is done using Data Query Languages (DQL) which allow searching of any data in the database and performing computations on it.

### What is Persistent Storage?

Persistent Storage known as non-volatile storage; persistence storage refers to any of the data storage devices that can retain data even after there is no power supply to that device. Among some of the common types of persistent storage are magnetic media, such as hard disk drives, tapes and several forms of optical media such as DVD. Persistent storage structures typically can be in the form of storage for files, blocks or objects.

### Benefits of Persistent Storage

- **Simplicity:**

Persistent storage helps developers provision their storage, without necessarily needing any expertise as storage experts. It simply allows them to provision volumes for both on-premise/ public cloud services.

- **Security:**

When it comes to the security and encryption aspects of storage solutions, persistent storage scores high. It fulfils the security requirements of most enterprises in terms of volume-level encryption, self-encrypting disks, and key management, among others to protect them against any kind of data loss and security breaches.

- **Flexibility:**

Persistent storage offers you great flexibility over traditional storage and lets you use the same software across different virtual machines, clouds and containers. Further, developers also enjoy the flexibility to choose the storage interfaces for their workload, including file, block or object storage. It also gives developers the ability to deliver data services with one system, irrespective of protocol, thus boosting productivity, offering more freedom, and leading to more effective application development.

- **Portability:**

Today's cloud-native world requires organizations to adopt a hybrid cloud approach to be able to combine the benefits of public and on-premises clouds. Persistent storage makes it easy to migrate your stateless applications across multiple clouds and migrate your data from one cloud to other clouds.

- **Efficiency:**

Persistent storage makes application development much more efficient. It eliminates the need to rewrite applications when you want to port them from one cloud provider to another and you can simply move applications without expensive or time-consuming rewrites whenever you want.

- **Cost-effectiveness:**

With persistent storage, you only have to pay for the storage and compute you use. It scales on-demand with no disruptions, growing and shrinking automatically as you add and remove files.

### What is MySQL?

MySQL is world's most popular database that is open source and free. MySQL was acquired by Oracle as a part of Sun Microsystems acquisition in 2009. MySQL is one of the most popular database management systems. It is an important pillar of LAMP application software. Further, it is cross-platform so works well with all operating systems including Windows, Linux, macOS, IRIX, and others. In addition, MySQL is a database management system used to manage relational databases. MySQL DBMS is owned by Oracle but still, it is open-source software and you can use it without paying anything. Moreover, you can change its source code according to your requirements and needs. There is comprehensive documentation available regarding development and deployment.

### Why MySQL is so popular?

MySQL is one of the most popular open-source DBMS and the following are the reasons behind it:

- It is completely free and open-source database management system.
- MySQL database is very user friendly mostly used with PHP, the most popular language for web development.
- It is a very powerful DBMS that offers a large set of functionalities which is only offered by most expensive DBMS.
- MySQL is cross platform and works with almost all operating systems.
- It is customizable because it is an open-source database. Developers can edit code according to their needs.
- It is quicker than most of other open-source databases and works well and fast even with the large data set.
- It is compatible with many languages like PHP, PERL, C, C++, JAVA, etc.
- There are very good stats on its Github repo.

### Features of MySQL

- **Speed:** Of course, the speed at which a server-side program runs depends primarily on the server hardware. Given that the server hardware is optimal, MySQL runs very fast. It supports clustered servers for demanding applications.
- **Ease of use:** MySQL is a high-performance, relatively simple database system. From the beginning, MySQL has typically been configured, monitored, and managed from the command line. However, several MySQL graphical interfaces are available as described below:
  - MySQL Administrator: This tool makes it possible for administrators to set up, evaluate, and tune their MySQL database server. This is intended as a replacement for mysqladmin.
  - MySQL Query Browser: Provides database developers and operators with a graphical database operation interface. It is especially useful for seeing multiple query plans and result sets in a single user interface.
  - Configuration Wizard: Administrators can choose from a predefined list of optimal settings, or create their own.
  - MySQL System Tray: Provides Windows-based administrators a single view of their MySQL instance, including the ability to start and stop their database servers.

- **Cost:** MySQL is available free of cost. MySQL is an "Open Source" database. MySQL is part of LAMP (Linux, Apache, MySQL, PHP / Perl / Python) environment, a fast-growing open-source enterprise software stack. More and more companies are using LAMP as an alternative to expensive proprietary software stacks because of its lower cost, reliability, and documentation.
- **Query Language Support:** MySQL understands standards-based SQL (Structured Query Language).
- **Capability:** Many clients can connect to the server at the same time. Clients can use multiple databases simultaneously. You can access MySQL using several interfaces such as command-line clients, Web browsers.
- **Connectivity and security:** MySQL are fully networked, and database can be accessed from anywhere on the Internet, so you can share your data with anyone, anywhere. The connectivity could be achieved with Windows programs by using ODBC drivers. By using the ODBC connector to MySQL, any ODBC-aware client application (for example, Microsoft Office, report writers, Visual Basic) can connect to MySQL.
- **Portability:** MySQL runs on many varieties of UNIX, as well as on other non-UNIX systems, such as Windows and OS/2. MySQL runs on hardware from home PCs to high-end server. MySQL can be installed on Windows XP, Windows Server 2003, Red Hat Fedora Linux, Debian Linux, and others.

## Benefits of MySQL

- **Flexible and easy to use:** Modify source code according to your own needs and expectations. The installation process is relatively simple, and doesn't take much time.
- **High performance:** Even if you are storing massive amounts of big e-Commerce data or doing heavy business intelligence activities, MySQL can assist you smoothly with optimum speed.
- **A mature DBMS:** Developers have been using MySQL for years, which means that there are abundant resources for them. It has evolved over the years and has very little margin for any kind of bugs.
- **Secure database:** Data security is the basic need of every web app. With its Access Privilege System and User Account Management, MySQL sets the security bar high. It offers both Host-based verification and password encryption as well.
- **Free installation:** The community edition of MySQL is free to download. There are some prepaid options but if your company is too small to pay for them, the free-to-download model is the most suitable for a fresh start.
- **Simple syntax:** MySQL's structure very simple and plain. That's why developers even consider MySQL a database with a human-like language. MySQL is easy to use, most of the tasks can be executed right in the command line, reducing development steps.

## Cons of MySQL

- **Owned by Oracle:** MySQL used to be open source but now it's not completely open source. It's mostly now under Oracle's license which limits the MySQL community in terms of improving the DBMS.
- **Scalability issues:** MySQL is not as efficiently scalable as the NoSQL database. It will need more engineering effort to make it scalable. So, if you have apps for which your database will increase substantially, you should choose another DBMS option.
- **Limited support for SQL standards:** MySQL doesn't completely comply with SQL standards. It does not provide support for some standard SQL features as well as it has some extensions and features that don't match the Structured Query Language standards.

## 3.2 Familiarizing with MySQL Commands

### Categories of SQL statements

SQL language is divided into four types of primary language statements: DML, DDL, DCL and TCL. Using these statements, we can define the structure of a database by creating and altering database objects and we can manipulate data in a table through updates or deletions. We also can control which user can read/write data or manage transactions to create a single unit of work.

The four main categories of SQL statements are as follows –

#### DML (Data Manipulation Language)

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following –

- SELECT – select records from a table
- INSERT – insert new records
- UPDATE – update/Modify existing records
- DELETE – delete existing records

#### DDL (Data Definition Language)

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

- CREATE – create a new Table, database, schema
- ALTER – alter existing table, column description
- DROP – delete existing objects from database

#### DCL (Data Control Language)

DCL statements control the level of access that users have on database objects.

- GRANT – allows users to read/write on certain database objects
- REVOKE – keeps users from read/write permission on database objects

#### TCL (Transaction Control Language)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

- BEGIN Transaction – opens a transaction
- COMMIT Transaction – commits a transaction

- ROLLBACK Transaction – ROLLBACK a transaction in case of any error

Lets understand SQL commands in detail.

### **DDL Commands - Data Definition Language**

Data Definition Language or DDL commands in SQL are used for changing the structure of a table. In other words, DDL commands are capable of creating, deleting, and modifying data. All DDL commands are auto-committed which means that changes made by them are automatically saved in the database. Following are the various DDL commands:

#### **ALTER**

Used for altering the structure of a database. Typically, the ALTER command is used either to add a new attribute or modify the characteristics of some existing attribute.

For adding new columns to the table:

General Syntax

```
ALTER TABLE table_name ADD (column_name1 data_type (size), column_name2 data_type (size),....., column_nameN data_type (size));
```

Example

```
ALTER TABLE Student ADD (Address varchar2(20));
ALTER TABLE Student ADD (Age number(2), Marks number(3));
```

For modifying an existing column in the table:

General Syntax:

```
ALTER TABLE table_name MODIFY (column_name new_data_type(new_size));
```

Example:

```
ALTER TABLE Student MODIFY (Name varchar2(20));
```

The ALTER command can also be used for dropping a column from the table:

General Syntax:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Student DROP COLUMN Age;
</pre>
```

**Note:** - It **is not** possible to **do** the following using the ALTER command:

- Change the name of a column
- Change the name of a table
- Decrease the size of a column

#### **CREATE**

Used for creating a new table in the database. General Syntax:

```
CREATE TABLE table_name (column_name1 data_type(size), column_name2 data_type(size),....., column_nameN data_type(size));
```

Example:

```
CREATE TABLE Employee(Name varchar2(20), D.O.B. date, Salary number(6);
```

#### DROP

Used for deleting an entire table from the database and all the data stored in it.

General Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Student;
```

#### RENAME

Used for renaming a table.

General Syntax:

```
RENAME old_table_name TO new_table_name
```

Example:

```
RENAME Student TO Student_Details
```

#### TRUNCATE

Used for deleting all rows from a table and free the space containing the table.

General Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Student;
```

### DML Commands - Data Manipulation Language

The DML or Data Manipulation Language commands help in modifying a relational database. These commands are not auto-committed, which simply means that all changes made to the database using DML commands aren't automatically saved.

It is possible to rollback DML commands. Various DML commands are:

#### DELETE

Used for removing one or more rows from a table.

General Syntax:

```
DELETE FROM table_name; (deletes all rows from a table)
```

```
DELETE FROM table_name WHERE some_condition; (delete only the row(s) where the condition is true)
```

Example:

```
DELETE FROM Student;
```

```
DELETE FROM Student WHERE Name = "Akhil";
```

#### INSERT

Used for inserting data into the row of a table.

General Syntax:

```
INSERT INTO table_name (column_name1, column_name2,...,column_nameN)
VALUES (value1, value2,...,valueN);
OR
INSERT INTO table_name VALUES (value1, value2,...,valueN);
```

Example:

```
INSERT INTO Student (Name, Age) VALUES ("Vijay", "25");
```

Insert command can also be used for inserting data into a table from another table.  
General Syntax:

```
INSERT INTO table_name1 SELECT column_name1,
column_name2,...,column_nameN FROM table_name2;
```

Example:

```
INSERT INTO Student SELECT Id, Stream FROM Student_Subject_Details
```

#### **UPDATE**

Used to modify or update the value of a column in a table. It can update all rows or some selective rows in the table.

General Syntax:

```
UPDATE table_name SET column_name1 = value1, column_name2 =
value2,...,column_nameN = valueN (for updating all rows)
UPDATE table_name SET column_name1 = value1, column_name2 =
value2,...,column_nameN = valueN [WHERE CONDITION] (for updating particular
rows)
```

Example:

```
UPDATE Student SET Name = "Akhil" WHERE Id = 22;
```

#### DCL Commands - Data Control Language

##### **GRANT**

Used for granting user access privileges to a database.

General Syntax:

```
GRANT object_privileges ON table_name TO user_name1,
user_name2,...,user_nameN;
GRANT object_privileges ON table_name TO user_name1,
user_name2,...,user_nameN WITH GRANT OPTION; (allows the grantee to grant
user access privileges to others)
```

Example:

```
GRANT SELECT, UPDATE ON Student TO Akhil Bhadwal
```

This will allow the user to run only SELECT and UPDATE operations on the Student table.

## **GRANT ALL ON Student TO Akhil Bhadwal WITH GRANT OPTION**

Allows the user to run all commands on the table as well as grant access privileges to other users.

### **REVOKE**

Used for taking back permission given to a user.

General Syntax:

```
REVOKE object_privileges ON table_name FROM user1, user2,... userN;
```

Example:

```
REVOKE UPDATE ON Student FROM Akhil;
```

**Note:** - A user who is not the owner of a table but has been given the privilege to grant permissions to other users can also revoke permissions.

### **TCL Commands - Transaction Control Language**

Transaction Control Language commands can only be used with DML commands. As these operations are auto-committed in the database, they can't be used while creating or dropping tables. Various TCL commands are:

#### **COMMIT**

Used for saving all transactions made to a database. Ends the current transaction and makes all changes permanent that were made during the transaction. Releases all transaction locks acquired on tables.

General Syntax:

```
COMMIT;
```

Example:

```
DELETE FROM Student WHERE Age = 25;
```

```
COMMIT;
```

#### **ROLLBACK**

Used to undo transactions that aren't yet saved in the database. Ends the transaction and undoes all changes made during the transaction. Releases all transaction locks acquired on tables.

General Syntax:

```
ROLLBACK;
```

Example:

```
DELETE FROM Student WHERE Age = 25;
```

```
ROLLBACK;
```

## 3.3 Connecting with Server

### What is JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as

- Java Applications
- Java Applets
- Java Servlets
- Java Server Pages (JSPs)
- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data. JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

### JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –

- JDBC API – This provides the application-to-JDBC Manager connection.
- JDBC Driver API – This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application. JDBC in Java uses a driver manager and a database-specific driver to connect to a database. The JDBC driver manager makes sure that the correct driver is being used to access the databases. It is also capable of handling multiple drivers connected to multiple databases simultaneously.

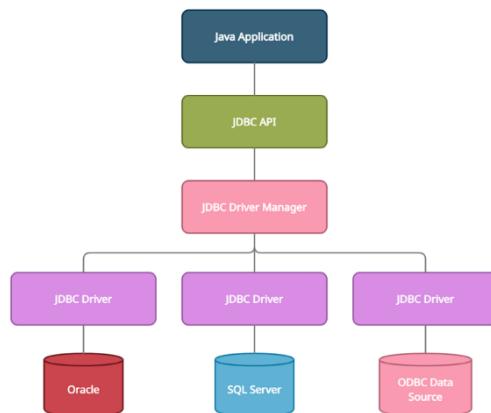


Image: JDBC Architecture

Reference: <https://usemynotes.com/wp-content/uploads/2021/07/jdbc-architecture-.png>

## Common JDBC Components

The JDBC API provides the following interfaces and classes

- **Driver Manager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use Driver Manager objects, which manage objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **Result Set:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

## JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers:

- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

## JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver. Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

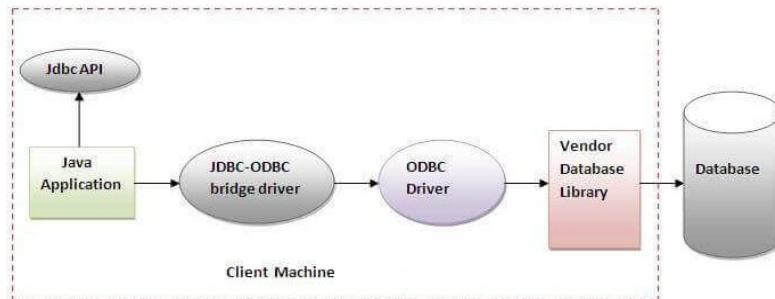


Image: JDBC-ODBC Bridge Driver

Reference: <https://www.javatpoint.com/jdbc-driver>

### Advantages:

- easy to use.
- can be easily connected to any database.

### Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

## Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

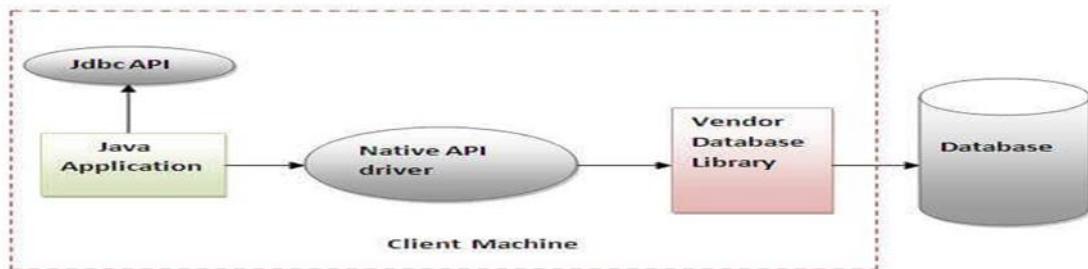


Image: Native-API Driver

Reference: <https://www.javatpoint.com/jdbc-driver>

### Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

### Disadvantage:

- The Native driver needs to be installed on each client machine.
- The Vendor client library needs to be installed on client machine.

## Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

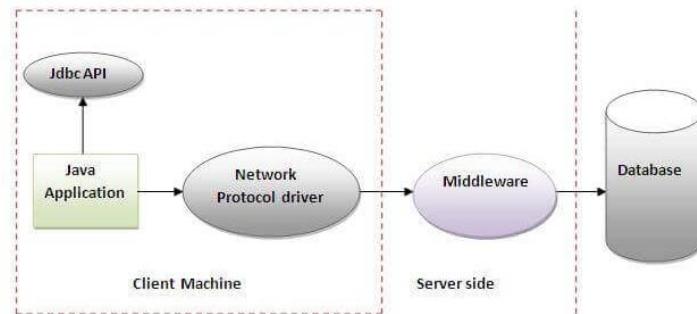


Image: Network Protocol Driver

Reference: <https://www.javatpoint.com/jdbc-driver>

#### **Advantage:**

- No client-side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

#### **Disadvantages:**

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

#### **Thin driver**

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

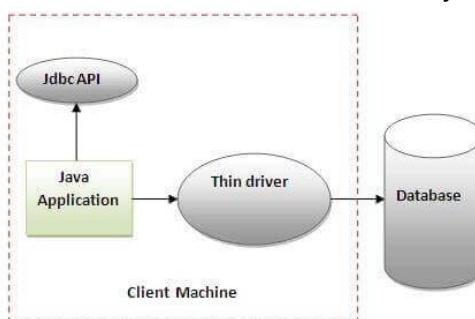


Image: Thin Driver

Reference: <https://www.javatpoint.com/jdbc-driver>

#### **Advantage:**

- Better performance than all other drivers.
- No software is required at client side or server side.

#### **Disadvantage:**

- Drivers depend on the Database.

### **Java Database Connectivity with 5 Steps**

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class

- Create connection
- Create statement
- Execute queries
- Close connection

### Register the driver class

The `forName()` method of `Class` class is used to register the driver class. This method is used to dynamically load the driver class.

#### Syntax of `forName()` method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register mysqlDriver class

```
import java.sql.DriverManager;
public class RegisterMysqlDriver {
 public static void main(String[] args) throws Exception {
 // Load the MysqlDriver class.
 Class.forName("com.mysql.cj.jdbc.Driver");
 // Register the MysqlDriver class with the DriverManager.
 DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());
 }
}
```

### Create the connection object

The `getConnection()` method of `DriverManager` class is used to establish connection with the database.

#### Syntax of `getConnection()` method

```
public static Connection getConnection(String url)throws SQLException
public static Connection getConnection(String url, String name, String password)
throws SQLException
```

Example to establish connection with the Mysql database

```
import java.sql.*;

public class ConnectMysql {

 public static void main(String[] args) throws Exception {
 // Load the MysqlDriver class.
 Class.forName("com.mysql.cj.jdbc.Driver");

 // Create a connection to the database.
 String url = "jdbc:mysql://localhost:3306/test";
 String username = "root";
 String password = "password";
 Connection connection = DriverManager.getConnection(url, username, password);

 // Do something with the connection.
 Statement statement = connection.createStatement();
 ResultSet resultSet = statement.executeQuery("SELECT * FROM users");
 while (resultSet.next()) {
 System.out.println(resultSet.getString("username"));
 }
 }
}
```

```
// Close the connection.
connection.close();
}
}
```

### Create the Statement object

The `createStatement()` method of `Connection` interface is used to create statement. The object of statement is responsible to execute queries with the database.

#### Syntax of `createStatement()` method

```
public Statement createStatement()throws SQLException
```

### Example to create the statement object

```
Statement stmt=con.createStatement();
```

### Execute the query

The `executeQuery()` method of `Statement` interface is used to execute queries to the database. This method returns the object of `ResultSet` that can be used to get all the records of a table.

#### Syntax of `executeQuery()` method

```
public ResultSet executeQuery(String sql)throws SQLException
```

### Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

### Close the connection object

By closing connection object statement and `ResultSet` will be closed automatically. The `close ()` method of `Connection` interface is used to close the connection.

#### Syntax of `close()` method

```
public void close()throws SQLException
```

### Example to close connection

```
con.close();
```

## Java JDBC CRUD: SQL Insert, Select, Update, and Delete

### 1. Prerequisites

To begin, make sure you have the following pieces of software installed on your computer:

- JDK ([download JDK 7](#)).
- MySQL ([download MySQL Community Server 5.6.12](#)). You may also want to [download MySQL Workbench](#) - a graphical tool for working with MySQL databases.
- JDBC Driver for MySQL ([download MySQL Connector/J 5.1.25](#)). Extract the zip archive and put the `mysql-connector-java-VERSION-bin.jar` file into classpath (in a same folder as your Java source files).

## 2. Creating a sample MySQL database

Let's create a MySQL database called *SampleDB* with one table *Users* with the following structure:

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
user_id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>					
username	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
password	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
fullname	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
email	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				

Execute the following SQL script inside MySQL Workbench:  
create database SampleDB;

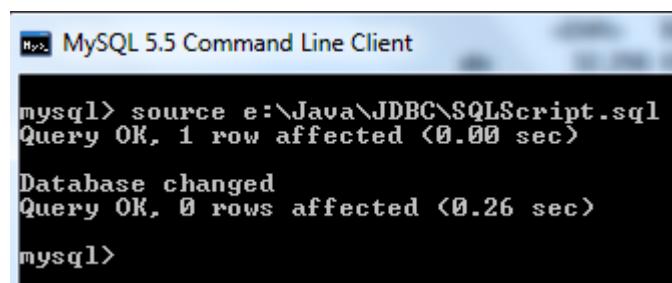
```
use SampleDB;
```

```
CREATE TABLE `users` (
 `user_id` int(11) NOT NULL AUTO_INCREMENT,
 `username` varchar(45) NOT NULL,
 `password` varchar(45) NOT NULL,
 `fullname` varchar(45) NOT NULL,
 `email` varchar(45) NOT NULL,
 PRIMARY KEY (`user_id`)
);
```

Or if you are using *MySQL Command Line Client* program, save the above script into a file, let's say, **SQLScript.sql** and execute the following command:

**source Path\To\The\Script\File\SQLScript.sql**

Here's an example screenshot taken while executing the above script in *MySQL Command Line Client* program:



The screenshot shows the MySQL 5.5 Command Line Client window. The command prompt is at the bottom, showing "mysql>". The main area displays the output of the "source" command:

```
mysql> source e:\Java\JDBC\SQLScript.sql
Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.26 sec)

mysql>
```

## 3. Understand the main JDBC interfaces and classes

Let's take an overview look at the JDBC's main interfaces and classes with which we usually work. They are all available under the **java.sql** package:

- **DriverManager**: this class is used to register driver for a specific database type (e.g. MySQL in this tutorial) and to establish a database connection with the server via its **getConnection()** method.

- **Connection:** this interface represents an established database connection (session) from which we can create statements to execute queries and retrieve results, get metadata about the database, close connection, etc.
- **Statement and PreparedStatement:** these interfaces are used to execute static SQL query and parameterized SQL query, respectively. Statement is the super interface of the PreparedStatement interface. Their commonly used methods are:
  - **boolean execute(String sql):** executes a general SQL statement. It returns true if the query returns a ResultSet, false if the query returns an update count or returns nothing. This method can be used with a Statement only.
  - **int executeUpdate(String sql):** executes an INSERT, UPDATE or DELETE statement and returns an update account indicating number of rows affected (e.g. 1 row inserted, or 2 rows updated, or 0 rows affected).
  - **ResultSet executeQuery(String sql):** executes a SELECT statement and returns a ResultSet object which contains results returned by the query.
- A prepared statement is one that contains placeholders (in form question marks ?) for dynamic values will be set at runtime. For example:
- **SELECT \* from Users WHERE user\_id=?**
- Here the value of user\_id is parameterized by a question mark and will be set by one of the setXXX() methods from the PreparedStatement interface, e.g. setInt(int index, int value).
- **ResultSet:** contains table data returned by a SELECT query. Use this object to iterate over rows in the result set using **next()** method, and get value of a column in the current row using **getXXX()** methods (e.g. **getString()**, **getInt()**, **getFloat()** and so on). The column value can be retrieved either by index number (1-based) or by column name.
- **SQLException:** this checked exception is declared to be thrown by all the above methods, so we have to catch this exception explicitly when calling the above classes' methods.

#### 4. Connecting to the database

Supposing the MySQL database server is listening on the default port 3306 at *localhost*. The following code snippet connects to the database name *SampleDB* by the user *root* and password *secret*.

```
String dbURL = "jdbc:mysql://localhost:3306/sampledb";
String username = "root";
String password = "secret";

try {
 Connection conn = DriverManager.getConnection(dbURL, username, password);
}
```

```
if (conn != null) {
 System.out.println("Connected");
}
} catch (SQLException ex) {
```

```
 ex.printStackTrace();
}
```

Once the connection was established, we have a Connection object which can be used to create statements in order to execute SQL queries. In the above code, we have to close the connection explicitly after finish working with the database:

```
conn.close();
```

### INSERT Statement

```
String sql = "INSERT INTO Users (username, password, fullname, email) VALUES (?, ?, ?, ?)";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");
statement.setString(2, "secretpass");
statement.setString(3, "Bill Gates");
statement.setString(4, "bill.gates@microsoft.com");

int rowsInserted = statement.executeUpdate();
if (rowsInserted > 0) {
 System.out.println("A new user was inserted successfully!");
}
```

### SELECT Statement

```
String sql = "SELECT * FROM Users";
Statement statement = conn.createStatement();
ResultSet result = statement.executeQuery(sql);

int count = 0;
while (result.next()){
 String name = result.getString(2);
 String pass = result.getString(3);
 String fullname = result.getString("fullname");
 String email = result.getString("email");

 String output = "User #%-d: %s - %s - %s - %s";
 System.out.println(String.format(output, ++count, name, pass, fullname, email));
}
```

### UPDATE Statement

```
String sql = "UPDATE Users SET password=?, fullname=?, email=? WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "123456789");
statement.setString(2, "William Henry Bill Gates");
statement.setString(3, "bill.gates@microsoft.com");
statement.setString(4, "bill");

int rowsUpdated = statement.executeUpdate();
if (rowsUpdated > 0) {
 System.out.println("An existing user was updated successfully!");
}
```

## **DELETE Statement**

```

String sql = "DELETE FROM Users WHERE username=?";

PreparedStatement statement = conn.prepareStatement(sql);
statement.setString(1, "bill");

int rowsDeleted = statement.executeUpdate();
if (rowsDeleted > 0) {
 System.out.println("A user was deleted successfully!");
}

```

## **Creating DAO (database access objects)**

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

- Data Access Object Interface - This interface defines the standard operations to be performed on a model object(s).
- Data Access Object concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- Model Object or Value Object - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

## **Implementation**

We are going to create a Student object acting as a Model or Value Object. StudentDao is Data Access Object Interface. StudentDaoImpl is concrete class implementing Data Access Object Interface. DaoPatternDemo, our demo class, will use StudentDao to demonstrate the use of Data Access Object pattern.

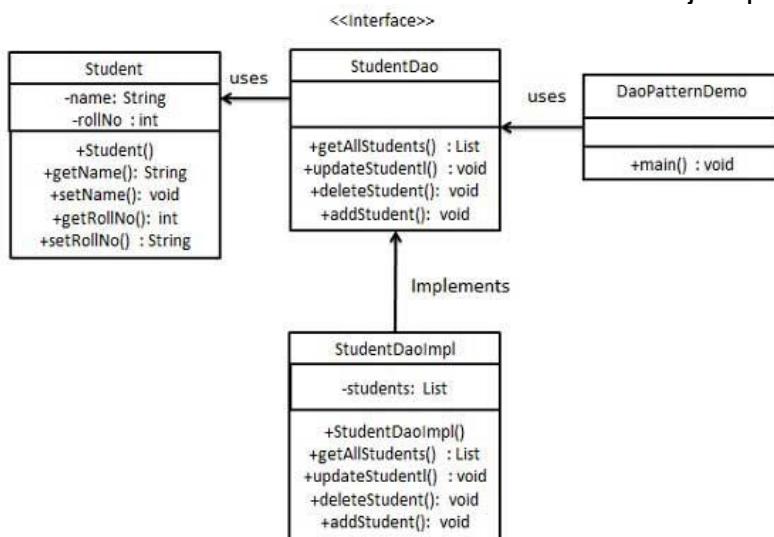


Image: DAO Implementation

Reference: [https://www.tutorialspoint.com/design\\_pattern/data\\_access\\_object\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm)

**Step 1:** Create Value Object.

### **Student.java**

```

public class Student {
 private String name;

```

```

private int rollNo;
Student(String name, int rollNo){
 this.name = name;
 this.rollNo = rollNo;
}

public String getName() {
 return name;
}

public void setName(String name) {
 this.name = name;
}

public int getRollNo() {
 return rollNo;
}

public void setRollNo(int rollNo) {
 this.rollNo = rollNo;
}

```

**Step 2:** Create Data Access Object Interface.

### StudentDao.java

```

import java.util.List;

public interface StudentDao {
 public List<Student> getAllStudents();
 public Student getStudent(int rollNo);
 public void updateStudent(Student student);
 public void deleteStudent(Student student);
}

```

**Step 3:** Create concrete class implementing above interface.

### StudentDaoImpl.java

```

import java.util.ArrayList;
import java.util.List;

public class StudentDaoImpl implements StudentDao {

 //list is working as a database
 List<Student> students;

 public StudentDaoImpl(){
 students = new ArrayList<Student>();
 }
}

```

```

Student student1 = new Student("Robert",0);
Student student2 = new Student("John",1);
students.add(student1);
students.add(student2);
}

@Override
public void deleteStudent(Student student) {
 students.remove(student.getRollNo());
 System.out.println("Student: Roll No " + student.getRollNo() + ", deleted from
database");
}

//retrieve list of students from the database
@Override
public List<Student> getAllStudents() {

 return students;
}

@Override
public Student getStudent(int rollNo) {
 return students.get(rollNo);
}

@Override
public void updateStudent(Student student) {
 students.get(student.getRollNo()).setName(student.getName());
 System.out.println("Student: Roll No " + student.getRollNo() + ", updated in the
database");
}
}

```

**Step 4:** Use the *StudentDao* to demonstrate Data Access Object pattern usage.

#### DaoPatternDemo.java

```

public class DaoPatternDemo {
 public static void main(String[] args) {
 StudentDao studentDao = new StudentDaoImpl();

 //print all students
 for (Student student : studentDao.getAllStudents()) {
 System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " +
student.getName() + "]");
 }

 //update student
 Student student = studentDao.getAllStudents().get(0);
 student.setName("Michael");
 studentDao.updateStudent(student);
 }
}

```

```

//get the student
studentDao.getStudent(0);
System.out.println("Student: [RollNo : " + student.getRollNo() + ", Name : " +
student.getName() + "]");
}
}

```

**Step 5:** Verify the output.

```

Student: [RollNo : 0, Name : Robert]
Student: [RollNo : 1, Name : John]
Student: Roll No 0, updated in the database
Student: [RollNo : 0, Name : Michael]

```

### Advantages

- The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently.
- If we need to change the underlying persistence mechanism, we only have to change the DAO layer and not all the places in the domain logic where the DAO layer is used from.

### Disadvantages

- Potential disadvantages of using DAO is a leaky abstraction, code duplication, and abstraction inversion.

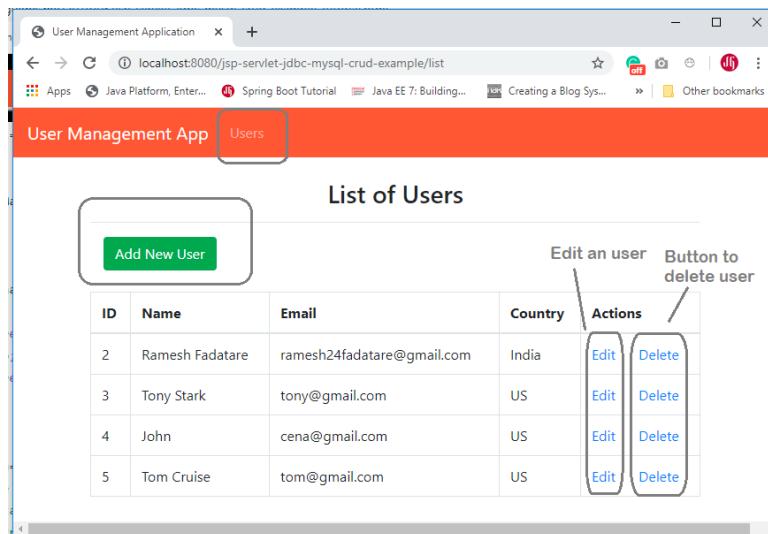
## 3.4 Managing data in databases using JSP & Servlet

### JSP Servlet JDBC MySQL CRUD

We will develop below simple basic features in our **User Management** web application:

1. Create a User
2. Update a User
3. Delete a User
4. Retrieve a User
5. List of all Users

The application looks something like this:



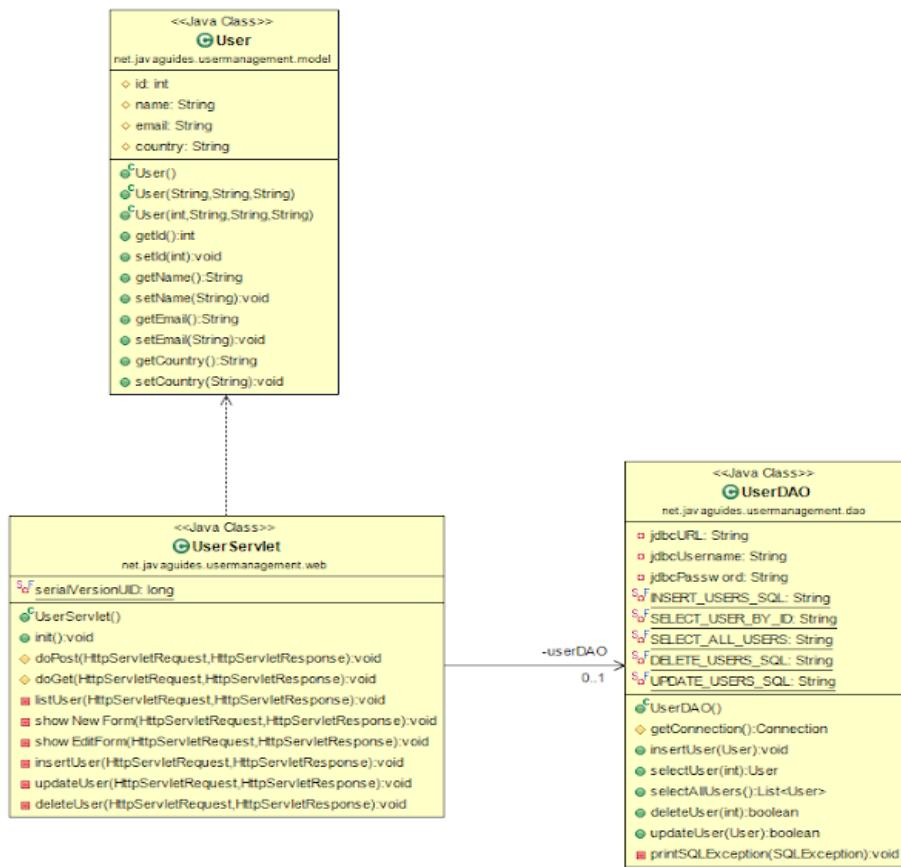
## Tools and Technologies used

- JSP - 2.2 +
- IDE - STS/Eclipse Neon.3
- JDK - 1.8 or later
- Apache Tomcat - 8.5
- JSTL - 1.2.1
- Servlet API - 2.5
- MySQL - mysql-connector-java-8.0.13.jar

## Development Steps

1. Create an Eclipse Dynamic Web Project
2. Add Dependencies
3. Project Structure
4. MySQL Database Setup
5. Create a JavaBean - User.java
6. Create a UserDao.java
7. Create a UserServlet.java
8. Creating User Listing JSP Page - user-list.jsp
9. Create a User Form JSP Page - user-form.jsp
10. Creating Error JSP page
11. Deploying and Testing the Application Demo

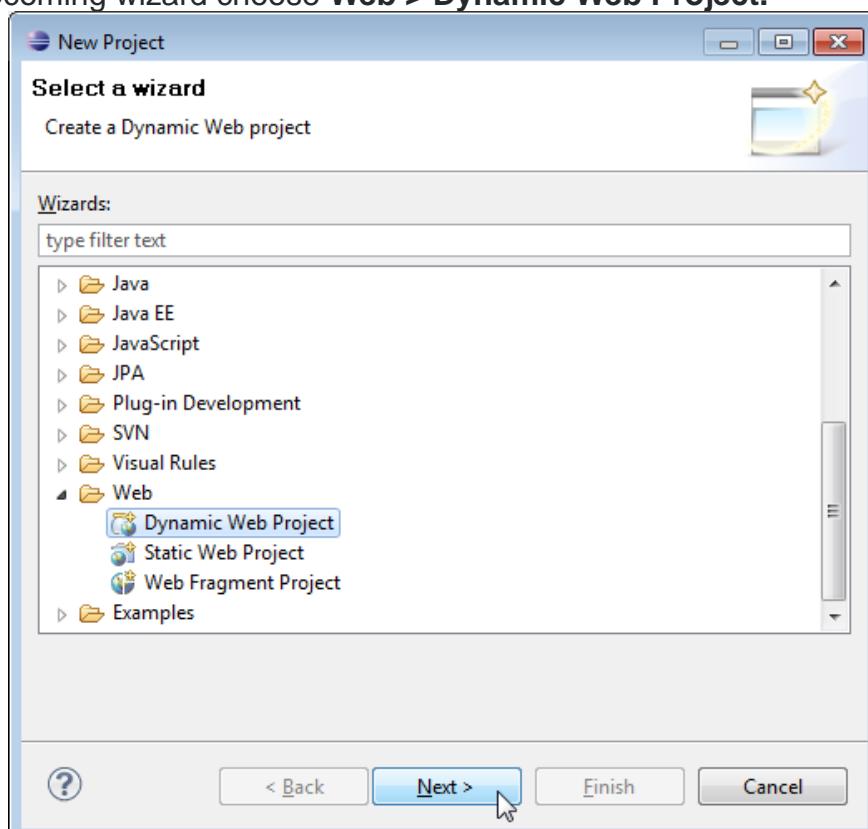
## Class Diagram



## 1. Create an Eclipse Dynamic Web Project

To create a new dynamic Web project in Eclipse:

1. On the main menu select **File > New > Project....**
2. In the upcoming wizard choose **Web > Dynamic Web Project.**



3. Click **Next**.
4. Enter project name as "jsp-servlet-jdbc-mysql-example";
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

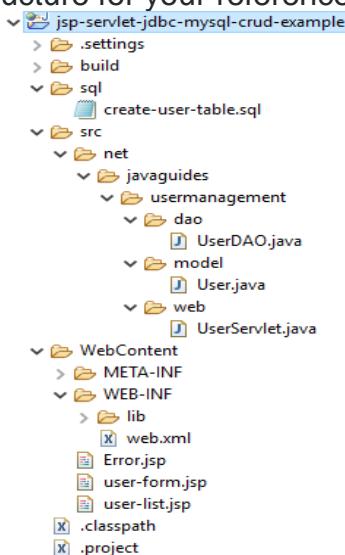
## 2. Add Dependencies

Add the latest release of below jar files to the **lib** folder.

- jsp-api.2.3.1.jar
- servlet-api.2.3.jar
- mysql-connector-java-8.0.13.jar
- jstl-1.2.jar

## 3. Project Structure

Standard project structure for your reference -



## 4. MySQL Database Setup

Let's create a database named "demo" in MySQL. Now, create a user's table using below DDL script:

```

CREATE DATABASE 'demo';
USE demo;

create table users (
 id int(3) NOT NULL AUTO_INCREMENT,
 name varchar(120) NOT NULL,
 email varchar(220) NOT NULL,
 country varchar(120),
 PRIMARY KEY (id)
);

```

You can use either MySQL Command Line Client or MySQL Workbench tool to create the database. The above a **user's** table looks like:

```

mysql> desc users;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id | int(3) | NO | PRI | NULL | auto_increment |
| name | varchar(120)| NO | | NULL | |
| email | varchar(220)| NO | | NULL | |
| country | varchar(120)| YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

## 5. Create a JavaBean - User.java

Let's create a `User` java class to model a `user` entity in the database with the following code:

```

package net.javaguides.usermanagement.model;

public class User {
 protected int id;
 protected String name;
 protected String email;
 protected String country;

 public User() {}

 public User(String name, String email, String country) {
 super();
 this.name = name;
 this.email = email;
 this.country = country;
 }

 public User(int id, String name, String email, String country) {
 super();
 this.id = id;
 this.name = name;
 this.email = email;
 this.country = country;
 }

 public int getId() {
 return id;
 }
 public void setId(int id) {
 this.id = id;
 }
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public String getEmail() {
 return email;
 }
}

```

```

 }
 public void setEmail(String email) {
 this.email = email;
 }
 public String getCountry() {
 return country;
 }
 public void setCountry(String country) {
 this.country = country;
 }
}

```

## 6. Create a UserDAO.java

Let's create a *UserDAO* class which is a Data Access Layer (DAO) class that provides CRUD (Create, Read, Update, Delete) operations for the table **users** in a database. Here's the full source code of the *UserDAO*:

```

package net.javaguides.usermanagement.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import net.javaguides.usermanagement.model.User;

public class UserDAO {
 private String jdbcURL = "jdbc:mysql://localhost:3306/demo?useSSL=false";
 private String jdbcUsername = "root";
 private String jdbcPassword = "root";

 private static final String INSERT_USERS_SQL = "INSERT INTO users" + "(name, email, country) VALUES " +
 " (?, ?, ?);"

 private static final String SELECT_USER_BY_ID = "select id,name,email,country from users where id =?";
 private static final String SELECT_ALL_USERS = "select * from users"
}

```

```

private static final String DELETE_USERS_SQL = "delete from users where id = ?;";
private static final String UPDATE_USERS_SQL = "update users set name = ?,email= ?, country =? where id = ?;"

public UserDAO() {}

protected Connection getConnection() {
 Connection connection = null;
}

```

```

try {
 Class.forName("com.mysql.jdbc.Driver");

 connection = DriverManager.getConnection(jdbcURL, jdbcUsername,
jdbcPassword);
} catch (SQLException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
} catch (ClassNotFoundException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
}
return connection;
}

public void insertUser(User user) throws SQLException {
 System.out.println(INSERT_USERS_SQL);
 // try-with-resource statement will auto close the connection.
 try (Connection connection = getConnection(); PreparedStatement preparedStatement = connection.prepareStatement(INSERT_USERS_SQL)) {
 preparedStatement.setString(1, user.getName());
 preparedStatement.setString(2, user.getEmail());
 preparedStatement.setString(3, user.getCountry());
 System.out.println(preparedStatement);
 preparedStatement.executeUpdate();
 } catch (SQLException e) {
 printSQLException(e);
 }
}

public User selectUser(int id) {
 User user = null;
 // Step 1: Establishing a Connection
 try (Connection connection = getConnection();
 // Step 2:Create a statement using connection object
 PreparedStatement preparedStatement = connection.prepareStatement(SELECT_USER_BY_ID);) {
 preparedStatement.setInt(1, id);
 System.out.println(preparedStatement);
 // Step 3: Execute the query or update query
 ResultSet rs = preparedStatement.executeQuery();

 // Step 4: Process the ResultSet object.
 while (rs.next()) {
 String name = rs.getString("name");
 String email = rs.getString("email");
 String country = rs.getString("country");
 user = new User(id, name, email, country);
 }
 } catch (SQLException e) {
 printSQLException(e);
 }
}

```

```
 return user; }
```

```
public List < User > selectAllUsers() {

 // using try-with-resources to avoid closing resources (boiler plate code)
 List < User > users = new ArrayList < > ();
 // Step 1: Establishing a Connection
 try (Connection connection = getConnection();

 // Step 2:Create a statement using connection object
 PreparedStatement preparedStatement =
connection.prepareStatement(SELECT_ALL_USERS);) {
 System.out.println(preparedStatement);
 // Step 3: Execute the query or update query
 ResultSet rs = preparedStatement.executeQuery();

 // Step 4: Process the ResultSet object.
 while (rs.next()) {
 int id = rs.getInt("id");
 String name = rs.getString("name");
 String email = rs.getString("email");
 String country = rs.getString("country");
 users.add(new User(id, name, email, country));
 }
 } catch (SQLException e) {
 printSQLException(e);
 }
 return users;
}

public boolean deleteUser(int id) throws SQLException {
 boolean rowDeleted;
 try (Connection connection = getConnection(); PreparedStatement statement =
connection.prepareStatement(DELETE_USERS_SQL);) {
 statement.setInt(1, id);
 rowDeleted = statement.executeUpdate() > 0;
 }
 return rowDeleted;
}

public boolean updateUser(User user) throws SQLException {
 boolean rowUpdated;
 try (Connection connection = getConnection(); PreparedStatement statement =
connection.prepareStatement(UPDATE_USERS_SQL);) {
 statement.setString(1, user.getName());
 statement.setString(2, user.getEmail());
 statement.setString(3, user.getCountry());
 statement.setInt(4, user.getId());

 rowUpdated = statement.executeUpdate() > 0;
 }
```

```

 }
 return rowUpdated;
 }

private void printSQLException(SQLException ex) {
 for (Throwable e: ex) {
 if (e instanceof SQLException) {
 e.printStackTrace(System.err);
 System.err.println("SQLState: " + ((SQLException) e).getSQLState());
 System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
 System.err.println("Message: " + e.getMessage());
 Throwable t = ex.getCause();
 while (t != null) {
 System.out.println("Cause: " + t);
 t = t.getCause();
 }
 }
 }
}

```

## 7. Create a UserServlet.java

Now, let's create *UserServlet* that acts as a page controller to handle all requests from the client. Let's look at the code first:

```

package net.javaguides.usermanagement.web;

import java.io.IOException;
import java.sql.SQLException;
import java.util.List;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import net.javaguides.usermanagement.dao.UserDAO;
import net.javaguides.usermanagement.model.User;

@WebServlet("/")
public class UserServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 private UserDAO userDAO;

 public void init() {
 userDAO = new UserDAO();
 }
}

```

```

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
 doGet(request, response);
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
 String action = request.getServletPath();

 try {
 switch (action) {
 case "/new":
 showNewForm(request, response);
 break;
 case "/insert":
 insertUser(request, response);
 break;
 case "/delete":
 deleteUser(request, response);
 break;
 case "/edit":
 showEditForm(request, response);
 break;
 case "/update":
 updateUser(request, response);
 break;
 default:
 listUser(request, response);
 break;
 }
 } catch (SQLException ex) {
 throw new ServletException(ex);
 }
}

private void listUser(HttpServletRequest request, HttpServletResponse response)
throws SQLException, IOException, ServletException {
 List < User > listUser = userDAO.selectAllUsers();
 request.setAttribute("listUser", listUser);
 RequestDispatcher dispatcher = request.getRequestDispatcher("user-list.jsp");
 dispatcher.forward(request, response);
}

private void showNewForm(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

```

```

 RequestDispatcher dispatcher = request.getRequestDispatcher("user-
form.jsp");
 dispatcher.forward(request, response);
 }

 private void showEditForm(HttpServletRequest request, HttpServletResponse
response)
throws SQLException, ServletException, IOException {
 int id = Integer.parseInt(request.getParameter("id"));
 User existingUser = userDAO.selectUser(id);
 RequestDispatcher dispatcher = request.getRequestDispatcher("user-
form.jsp");
 request.setAttribute("user", existingUser);
 dispatcher.forward(request, response);

}

private void insertUser(HttpServletRequest request, HttpServletResponse
response)
throws SQLException, IOException {
 String name = request.getParameter("name");
 String email = request.getParameter("email");
 String country = request.getParameter("country");
 User newUser = new User(name, email, country);
 userDAO.insertUser(newUser);
 response.sendRedirect("list");
}

private void updateUser(HttpServletRequest request, HttpServletResponse
response)
throws SQLException, IOException {
 int id = Integer.parseInt(request.getParameter("id"));
 String name = request.getParameter("name");
 String email = request.getParameter("email");
 String country = request.getParameter("country");

 User book = new User(id, name, email, country);
 userDAO.updateUser(book);
 response.sendRedirect("list");
}

private void deleteUser(HttpServletRequest request, HttpServletResponse
response)
throws SQLException, IOException {
 int id = Integer.parseInt(request.getParameter("id"));
 userDAO.deleteUser(id);
 response.sendRedirect("list");
}

```

```
}
```

## 8. Creating User Listing JSP Page - user-list.jsp

Next, create a JSP page for displaying all **users** from the database. Let's create a *list-user.jsp* page under the **WebContent** directory in the project with the following code:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>

 <head>
 <title>User Management Application</title>
 <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
 </head>

 <body>

 <header>
 <nav class="navbar navbar-expand-md navbar-dark" style="background-color: tomato">
 <div>
 User Management App
 </div>

 <ul class="navbar-nav">
 <a href="<%=request.getContextPath()%>/list" class="nav-link">Users

 </nav>
 </header>

 <div class="row">
 <!-- <div class="alert alert-success" *ngIf='message'>{{message}}</div> -->
 </div>

 <div class="container">
 <h3 class="text-center">List of Users</h3>
 <hr>
 <div class="container text-left">
 <a href="<%=request.getContextPath()%>/new" class="btn btn-success">Add
 </div>
 </div>
 </body>

```

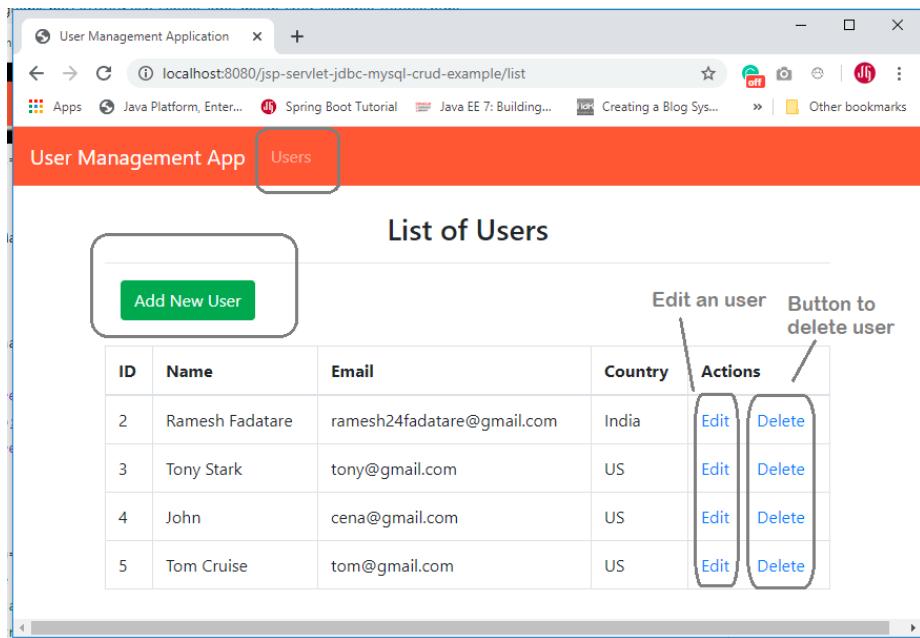
```
New User
 </div>

 <table class="table table-bordered">
 <thead>
 <tr>
 <th>ID</th>
 <th>Name</th>
 <th>Email</th>
 <th>Country</th>
 <th>Actions</th>
 </tr>
 </thead>
 <tbody>
 <!-- for (Todo todo: todos) { -->
 <c:forEach var="user" items="${listUser}">

 <tr>
 <td>
 <c:out value="${user.id}" />
 </td>
 <td>
 <c:out value="${user.name}" />
 </td>
 <td>
 <c:out value="${user.email}" />
 </td>
 <td>
 <c:out value="${user.country}" />
 </td>
 <td>
 <a href="edit?id=<c:out value='${user.id}' />">Edit
 ><a href="delete?id=<c:out value='${user.id}' />">Delete</td>
 </td>
 </tr>
 </c:forEach>
 <!-- } -->
 </tbody>
 </table>
 </div>
</div>
</body>

</html>
```

Once you will deploy above JSP page in tomcat and open in the browser looks something like this:



## 9. Create a User Form JSP Page - *user-form.jsp*

Next, we create a JSP page for creating a new User called *user-form.jsp*. Here's its full source code:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>

 <head>
 <title>User Management Application</title>
 <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous" rel="stylesheet">
 </head>

 <body>

 <header>
 <nav class="navbar navbar-expand-md navbar-dark" style="background-color: tomato">
 <div>
 User Management App
 </div>
 <ul class="navbar-nav">
 <a href="<%=request.getContextPath()%>/list" class="nav-link">Users
```

```


 </nav>
</header>

<div class="container col-md-5">
 <div class="card">
 <div class="card-body">
 <c:if test="${user != null}">
 <form action="update" method="post">
 </c:if>
 <c:if test="${user == null}">
 <form action="insert" method="post">
 </c:if>

 <caption>
 <h2>
 <c:if test="${user != null}">
 Edit User
 </c:if>
 <c:if test="${user == null}">
 Add New User
 </c:if>
 </h2>
 </caption>

 <c:if test="${user != null}">
 <input type="hidden" name="id" value=<c:out value='${user.id}' />">
 </c:if>

 <fieldset class="form-group">
 <label>User Name</label> <input type="text" value=<c:out value='${user.name}' />" class="form-control" name="name" required="required">
 </fieldset>

 <fieldset class="form-group">
 <label>User Email</label> <input type="text" value=<c:out value='${user.email}' />" class="form-control" name="email">
 </fieldset>

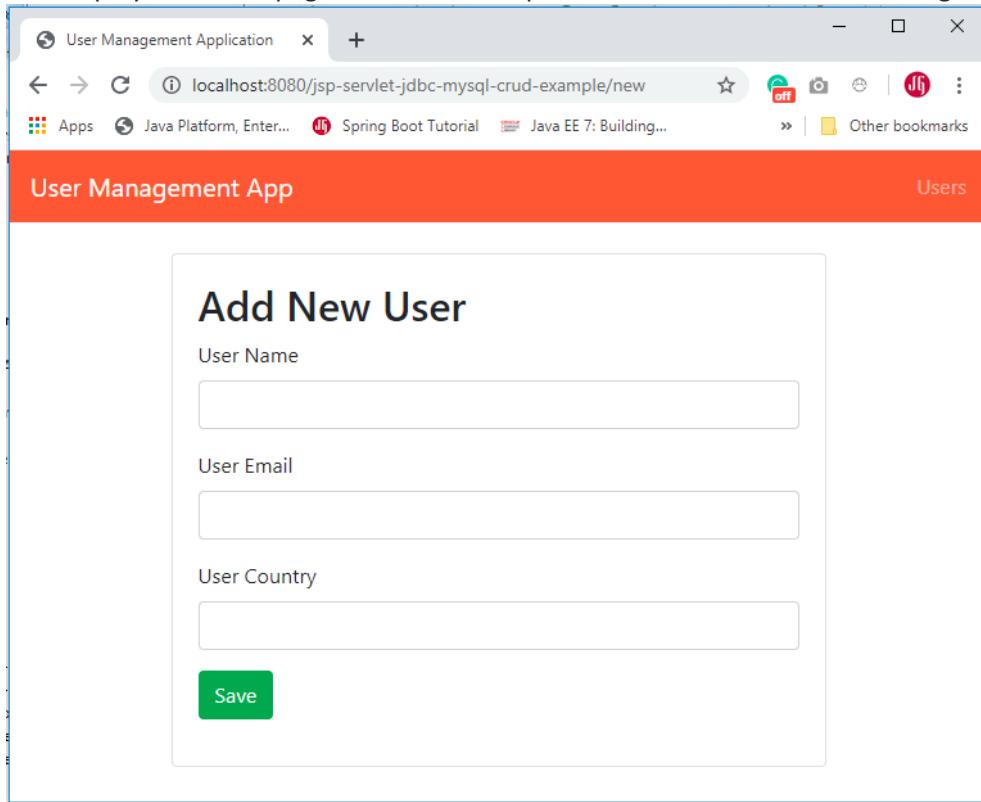
 <fieldset class="form-group">
 <label>User Country</label> <input type="text" value=<c:out value='${user.country}' />" class="form-control" name="country">
 </fieldset>

 <button type="submit" class="btn btn-success">Save</button>
 </form>
</div>
</div>
</div>

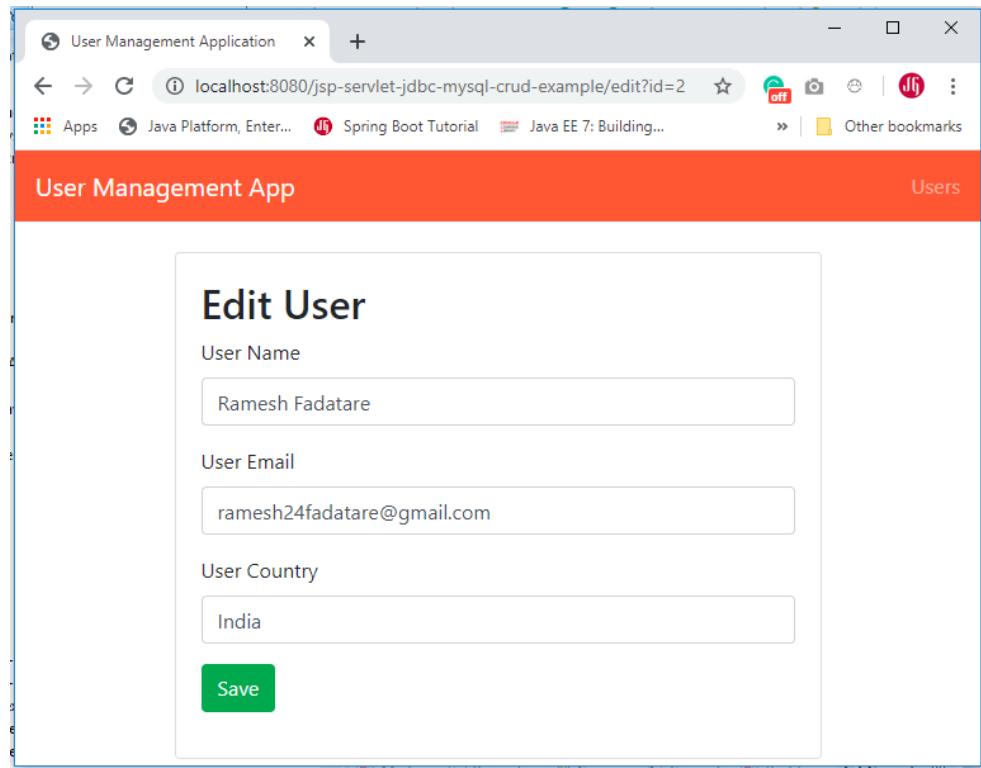
```

```
</body>
</html>
```

Once you will deploy above JSP page in tomcat and open in the browser looks something like this:



The above page acts for both functionalities to create a new User and Edit the same user. The edit page looks like:



## 10. Creating Error JSP page

Here's the code of the *Error.jsp* page which simply shows the exception message:

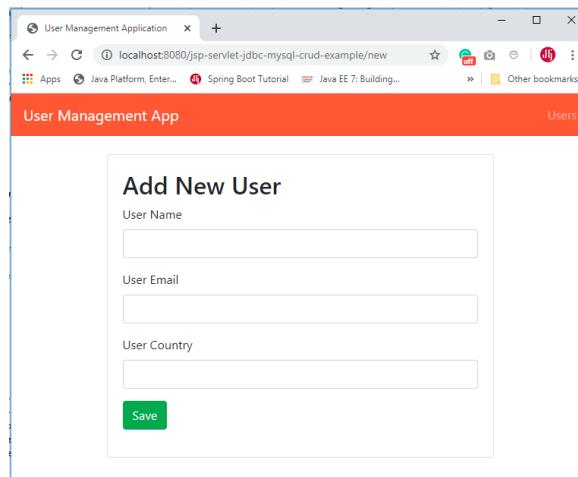
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
 pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Error</title>
</head>
<body>
<center>
<h1>Error</h1>
<h2><%=exception.getMessage() %>
 </h2>
</center>
</body>
</html>
```

## 11. Deploying and Testing the Application

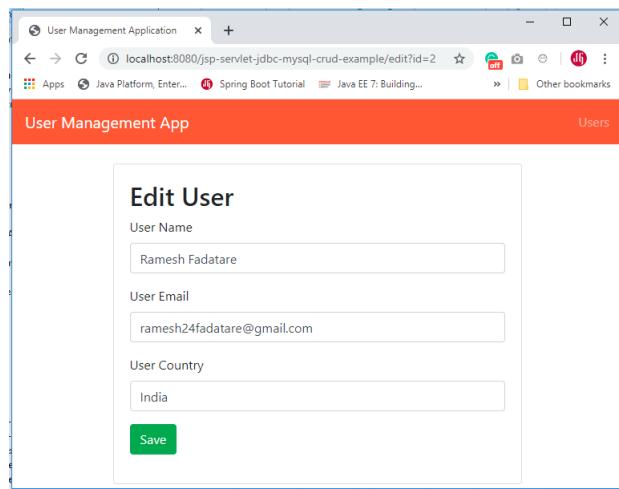
It's time to see a demo of the above **User Management** web application. Deploy this web application in tomcat server.

Type the following URL in your web browser to access the **User Management** application: <http://localhost:8080/jsp-servlet-jdbc-mysql-crud-example/>

### Create a new User



### Edit a User



### List of all Users

The screenshot shows a browser window titled "User Management Application" with the URL "localhost:8080/jsp-servlet-jdbc-mysql-crud-example/list". The page is titled "List of Users". At the top left is a green "Add New User" button. Below it is a table with columns: ID, Name, Email, Country, and Actions. The table has five rows of data:

ID	Name	Email	Country	Actions
2	Ramesh Fadatare	ramesh24fadatare@gmail.com	India	<a href="#">Edit</a> <a href="#">Delete</a>
3	Tony Stark	tony@gmail.com	US	<a href="#">Edit</a> <a href="#">Delete</a>
4	John	cena@gmail.com	US	<a href="#">Edit</a> <a href="#">Delete</a>
5	Tom Cruise	tom@gmail.com	US	<a href="#">Edit</a> <a href="#">Delete</a>

# Chapter 4: Getting Familiar with Restful API

### Learning Outcomes:

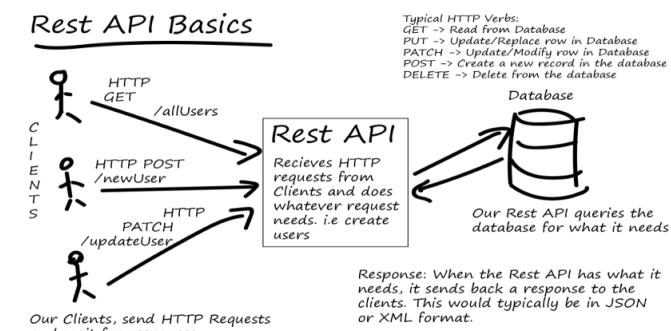
- Understanding RESTful Architecture
- Spring Boot Fundamentals
- Designing RESTful APIs
- Building CRUD Operations
- Exception Handling
- Testing and Documentation
- Performance Optimization and Caching
- Integration with Third-Party Services

## 5.1 Introduction

### 5.1.1 What is a REST API?

REST is a set of guidelines that software can use to communicate over the internet in order to make integrations simple and scalable. A REST API (also called a “RESTful” API) is a specific type of API that follows these guidelines.

REST stands for Representational State Transfer. This means that when a client requests a resource using a REST API, the server transfers back the current state of the resource in a standardized representation.



reference: <https://images.tutorialspoint.net/uploads/rest-api.png>

### 5.1.2 Why Do You Need a REST API?

#### 1. Simplifies Development

- If APIs didn't exist, developers would have to write a separate communication protocol for each third-party software or service with which they wanted to communicate. Not only would this be tremendously time-consuming and technically complex, but it would also be highly brittle—if one of the systems changed, it could break the entire protocol and force developers to start from scratch.
- Creating an API establishes a “lingua franca” for wildly divergent systems to speak to each other. Rather than having to learn each other's unique “language,” both entities can communicate in a common format, which can also be used by any other application or service.

#### 2. Provides an Abstraction for Technical Details

- Even if everyone spoke the same “language,” communicating between two different systems would still be technically challenging without an API. Developers would have to dive deep into low-level concepts such as data formats and data transfer, making it more challenging and laborious—and preventing you from getting quick, easy access to the information you need.

- APIs solve this problem by presenting a clean, smooth layer of abstraction over the messiness of the underlying technical implementation. Instead of working with low-level communications protocols, developers simply have to look up the syntax of how to send a high-level REST API request—a much easier task. With the right REST API, you don't need to understand the specifics of how the server stores or retrieves the data you're looking for. You only care about the results that it sends back in response to your request.

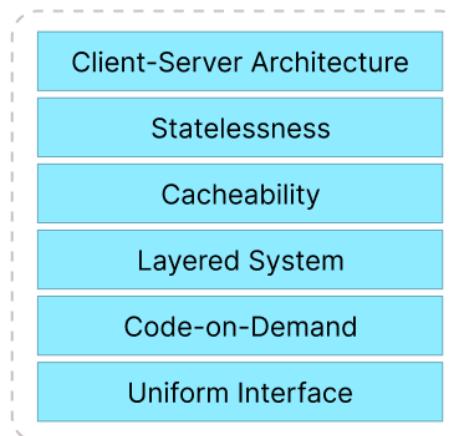
### 3. Most Popular API Architecture

- REST isn't the only API architecture out there. The older simple object access protocol (SOAP) architecture is still in use among a minority of organizations. However, REST is by far the predominant choice for building APIs. According to a 2017 report, 83% of APIs use the REST architecture, while 15% make use of SOAP. Meanwhile, just 2% rely on neither of these and instead use Microsoft's .NET architecture.
- The status of REST as the most popular API architecture carries with it a number of valuable benefits. For example, it's easier to find tools and tutorials for working with REST APIs. Tech giants such as Google, Amazon, Microsoft, and Twitter all actively promote REST APIs for accessing and using their services.

### 4. The technical benefits of REST APIs include

- REST is compatible with a variety of messaging formats, including XML, YAML, and JSON. However, SOAP only uses XML. This makes REST more flexible and lightweight for certain use cases.
- REST is much “leaner” than SOAP, without much overhead beyond the HTTP request itself. This makes the performance of REST slightly faster for servers, and much faster when traveling over the network. In contrast, it is necessary to wrap SOAP messages in an “envelope.”

#### 5.1.3 CHARACTERISTICS OF REST API



Reference :<https://www.scrapingbee.com/blog/six-characteristics-of-rest-api/#six-characteristics-of-rest>

## 1. Client-Server Architecture

- RESTful APIs are built with a client-server architecture, meaning that the client sends a request to the server and the server sends back a response. The client can be any device or application that can make HTTP requests, while the server is the application that provides the API and responds to client requests. This characteristic allows for the separation of concerns, making it easier to develop, maintain, and scale both of these components independently.

## 2. Statelessness

- RESTful APIs are stateless, meaning that each request made by the client to the server contains all the information necessary for the server to fulfill the request, without relying on any previous requests or server-side storage. This is why every authenticated REST request has to carry an authentication token in the request headers. This does increase the request size but it lets the server scale without worrying about storing state information across two separate requests.

## 3. Cache ability

- It is important to utilize methods to reduce the load on the server. Therefore, RESTful APIs implement some sort of caching. This means that the API responses can be cached by the client, allowing for faster response times in subsequent requests for the same resource. This reduces the load on the server and improves performance, as the server does not need to generate the same response for each request.
- As each request in REST has to carry authentication tokens and the relevant state required to act on the server, the benefits of this characteristic become visible pretty quickly for any API with decent traffic. Even with a cache timeout of 5s, you can help prevent thousands or millions of requests over a few seconds.

## 4. Layered System

- Future-proof APIs should be modular and each module should be updatable or swappable transparently. Hence, REST requires the APIs to be designed as a layered system, where the client interacts with the server through a single endpoint, while the server can interact with multiple backend systems. This provides a separation of concerns and makes it easier to add new backend systems, change existing ones, or perform maintenance, without affecting the client. An example of this might be how a server can update the mechanism for load-balancing but the client doesn't need to be made aware of that. The client can continue communication the same as before.

## 5. Code-On-Demand

- This is an optional characteristic as it can lead to unintended side effects and exploits. This characteristic means that the server can send back code to be executed by the client instead of data. This can help extend the functionality of the client and lead to more dynamic and customizable interactions. However, this also requires that the client can understand and execute the code that the server sends back. This has therefore reduced the frequency with which this characteristic is adhered to. Moreover, if the server is hacked, the clients will automatically be hijacked as they will execute whatever the server responds with. This glaring security gotcha has also hindered the adoption of Code-On-Demand.

## 6. Uniform Interface

- This means that the API uses a common set of methods, such as GET, POST, PUT, and DELETE, to access resources, and a standard format, such as JSON or XML, for requests and responses. This makes it easier for clients to understand and interact with the API, as all resources are accessed in a consistent manner. The uniform interface also makes it easier to implement API versioning, as new functionality can be added by defining new resources and methods, without affecting existing ones.

### The components of a RESTful API

RESTful APIs consist of the following components:

#### 1. The endpoints

- The endpoint describes the location of the data on the server. The endpoints are URLs of the resources you are trying to access through the API.

#### 2. The method

- We have already discussed the four HTTP methods (GET, PUT, POST, DELETE) APIs use to manipulate data. An API request must use one of these methods for the server to understand what needs to be done.

#### 3. The headers

- RESTful APIs contain HTTP headers that contain information, such as metadata, proxies, and HTTP connection types. In a request message, the header contains information on the nature of the request as well as the types of valid responses.
- In a response message, the header also contains information on the status of the request along with status codes. “404,” for example, means the API was unsuccessful at retrieving requested data from the server.

#### 4. The data (or body)

- The data (or body) of the RESTful API consists of further information on the resources requested by the client. In the case of a simple GET request, further information is not needed.

- In a POST request, the client will declare the type of content in the header, and the actual content will be in the body. This could be a new resource the client is submitting to the server. The server will see whether the content type is acceptable or not and proceed to find the resource in the body.

#### **5.1.4 Advantages of RESTful APIs**

RESTful APIs are fast, flexible, scalable, and versatile.

Here are some of the key advantages of this type of API:

##### **1. Supports all types of data formats**

- In other types of APIs, you are limited in your choice of data formats. However, RESTful APIs support all data formats.

##### **2. Works wonders with web browsers**

- In RESTful APIs, you can send an HTTP request, get JavaScript Object Notation (JSON) data in response, and parse that data to use it on the client applications. This makes it the best choice for web browsers. Furthermore, you can easily integrate these APIs with your existing website.

##### **3. Uses less bandwidth**

- Thanks to JSON, RESTful APIs use less bandwidth as compared to other types of APIs. However, this stands true for JSON-based web APIs only. An XML-based web API will have a similar payload to its non-RESTful counterpart whether it adheres to the REST architectural constraints or not.

##### **4. Does not need to be designed from scratch**

- In most cases, you can get models that you can modify and use. For example, NetApp and Mailgun provide a complete tutorial and source code for building a private API. In some cases, maybe when developing a private API, you will need to design the API from scratch, and for that, you can get a lot of support from Stack Overflow.

##### **5. Easy for most developers**

- RESTful APIs use HTTP methods for communication. You can use Python, JavaScript (Node.js), Ruby, C#, and a number of other languages to develop these APIs, making them easier to work with for most developers.

#### **5.1.5 RestAPI VS Restful API**

- ✓ The working principle of these two APIs is very much similar. Both of them allow you to access data from the server, but they differ in how they fetch and format data. The

RestFul API is more like a service where you get the data, whereas Rest API is more like a database where you place your query and get back results.

- ✓ The RESTful API is a set of resources that can be accessed via HTTP. The structure of the response is based on the reference resource. This means that each resource has its unique URI, and each call to the same help returns data with the same structure.
- ✓ The RESTful API is an alternative to the SOAP-based Web services that were prevalent in the past. It uses HTTP methods like GET, POST, PUT and DELETE to communicate with remote servers over the Internet. In addition, the RESTful API allows you to use your favorite programming language to write code for interacting with this type of service.
- ✓ The working principle of REST API is to use HTTP for all interactions with the server. The communication is usually encoded as JSON or XML with some content-type header in front of it. The server will return a representation of the data in this format, which can be parsed or transformed by the client into something else (e.g., into another form).
- ✓ REST API stands for Representational State Transfer API, and it's a platform-independent way to design and build web services that can be accessed using the HTTP protocol. With this approach, you can make your application work even if it runs on different platforms such as Android or iOS devices, Windows desktops, etc.

## 5.2 Moving Ahead with Spring Boot

### 5.2.1 Spring Boot

#### Introduction

- ✓ Spring Boot is an open-source Java-based framework developed by Pivotal Team and used to make stand-alone and production-ready spring applications. It used to make a micro-Service. It offers a decent platform for Java developers to build stand-alone spring application that you can just run. You can proceed with the least configurations without the need for a whole spring configuration setup.

#### Features of Spring Boot:

Spring Boot is built on the summit of the conservative spring framework. So, it offers all the features of spring and is yet easier to employ than spring.

##### 1. Auto-Configuration

- The problem that every developer has faced while increasing a plan is that of widespread configurations. Configurations for database, rest API, flyway, security etc. are obligatory in any applications.
- These can be sometimes extreme and time-consuming. But spring-boot decides these circumstances too. Spring Boot does the patterns routinely according to the dependencies added. For case, if we add spring-boot-starter-web addition to our project, then web server, Servlets etc. will be configured automatically.

##### 2. Component Scan

- One of the essential features of spring is reliance injection. The classes for auto-injection, spring uses particular typecast annotations such as @Component, @Controller, @Service, and @Repository. In order to make objects of these annotated classes and insert them, spring needs to know where they are situated.
- In spring boot application @SpringBootApplication annotation adds @ComponentScan annotation with non-payment setting. Hence by evasion spring boot scans the annotated classes under the present package.

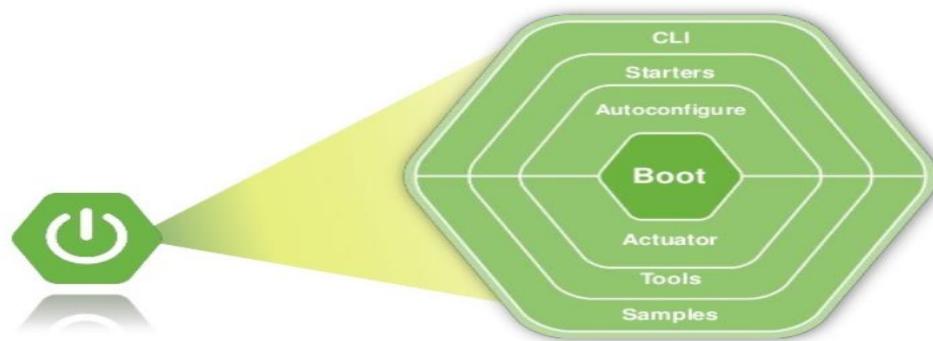
### 3. Auto dependency management

- Any project, even a separate project, is needy on some libraries for something. These libraries are obtainable as dependencies. Sometimes it occurs that even these libraries are reliant on some other libraries. Most of the times they require exact versions.
- If the connected dependency is missing or arrangements are not like-minded, the application does not work as expected. We cannot refute that managing the dependencies is tricky. Spring boot solves this difficulty by providing dependency-management section.

### 4. Externalized Configuration

- There are scenarios where some data used in an application is dissimilar for diverse environments. Such as, the port number, where the application is deployed is diverse for development and making surroundings. In such cases, hard coding them in the system could be tricky to direct and change.
- Spring boot permits numerous options to externalize these properties. We can employ these configured properties using the @Value footnote or by binding properties to objects using @Configuration Properties.

## Spring Boot Architecture Diagram And Components



Reference: <https://www.w3schools.blog/spring-boot-architecture-diagram-components>

**Spring Boot Starters:** The main concept behind spring boot starters is to reduce the dependencies definitions and to simplify the project build dependencies.

For example: if we are creating a spring application then we have to define the following dependencies in the pom.xml file –

- Spring core Jar file
- Spring Web Jar file
- Spring Web MVC Jar file
- Servlet Jar file

Spring Boot Starters provides the facility to add only one jar file spring-boot-starter-web instead of adding 4 jar files.

**Spring Boot AutoConfigurator:** Main concept behind Spring Boot AutoConfigurator is to minimizing the programmer's effort to define lots of XML configuration. It will take care of all these XML configurations and annotations.

**Spring Boot CLI:** It is Spring Boot software which is used to run and test Spring Boot applications from command prompt. CLI refers to command line arguments. To execute a spring application, spring boot CLI uses Spring Boot Starter and Spring Boot AutoConfigurate components to resolve all dependencies.

**Spring Boot Actuator:** Spring boot actuator is a tool which provides HTTP endpoints. We can manage our production application using these HTTP endpoints.

### Advantages & disadvantages of using spring boot

#### Advantages:

- Simplified & adaptation clash-free dependency management through the starter POMs.
- No XML based arrangements at all. Very much cut down properties. The beans are initialized, configured and wired automatically.
- Spring Boot offers HTTP endpoints to contact application internals like health status, detailed metrics, inner application working, etc.
- We can fast setup and run standalone, web applications and microservice at significantly fewer time.
- You can bring together the jar object which comes with an entrenched Tomcat, Jetty or Undertow application server and you are prepared to go.
- The Spring Initializer gives a project producer to make you creative with the convinced technology stack from the beginning.

#### Disadvantages:

- Spring boot may gratuitously augment the binary operation size with vacant dependencies.
- Spring Boot sticks well with microservice. The Spring Boot artefacts can be deployed straight into Docker containers.

## 5.2.2 Spring IO

### Overview

Spring IO is a cohesive, versioned platform for building modern applications. It is a modular, enterprise-grade distribution that delivers a curated set of dependencies while keeping developers in full control of deploying only the parts they need. Spring IO is 100% open source, lean, and modular.

The Spring IO platform includes Foundation Layer modules and Execution Layer domain-specific runtimes (DSRs). The Foundation layer represents the core Spring modules and associated third-party dependencies that have been harmonized to ensure a smooth development experience. The DSRs provided by the Spring IO Execution Layer dramatically simplify building production-ready, JVM-based workloads. The first release of Spring IO includes two DSRs: Spring Boot and Grails.

## Features

- One platform, many workloads - build web, integration, batch, reactive or big data applications
- Radically simplified development experience with Spring Boot
- Production-ready features provided out of the box
- Curated and harmonized dependencies that just work together
- Modular platform that allows developers to deploy only the parts they need
- Support for embedded runtimes, classic application server, and PaaS deployments
- Depends only on Java SE, and supports Groovy, Grails and some Java EE
- Works with your existing dependency management tools such as Maven and Gradle
- The Spring IO Platform is certified to work on JDK 7 and 8

## Using the Platform

The Spring IO platform provides versions of the various Spring projects and their dependencies. With the configuration shown above added to your build script, you're ready to declare your dependencies without having to worry about version numbers:

### Maven

```
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-core</artifactId>
</dependency>
```

### Gradle

```
dependencies {
 compile 'org.springframework:spring-core'
```

### 5.2.3 Using HTTP Methods prior RESTful Services

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

## Spring Boot JDBC

- JDBC ( Java Database Connectivity ) is a programming interface for the Java programming language that outlines how a client can access a database. It is a Java-based data access technology used to link Java databases.
- For connecting the application with JDBC spring boot, JDBC provides us with starters and libraries. Database-related beans such as DataSource, JdbcTemplate, and NamedParameterJdbcTemplate are auto-configured and produced during the Spring Boot JDBC startup. and if we want to use it, we can autowire following classes-

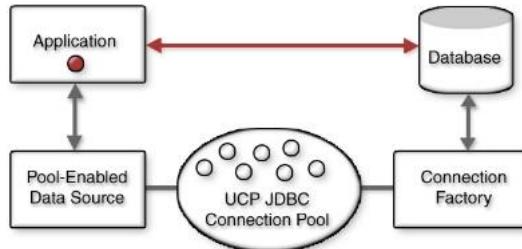
```

@Autowired
JdbcTemplate jdbcTemplate;
@Autowired
private NamedParameterJdbcTemplate jdbcTemplate;
```

We will configure DataSource and connection pooling present in the application.properties. By default, spring boot chooses tomcat pooling.

## What is JDBC connection pooling?

- Multiple database connection requests are managed by a mechanism called JDBC connection pooling. In other words, it promotes connection reuse by storing database connections in a memory cache known as a connection pool. It is maintained as a layer on top of any normal JDBC driver product by a connection pooling module.



Reference: <https://www.codingninjas.com/studio/library/spring-boot-jdbc>

- There are three clients in the above figure with a connection pool and a data source. Three customers are linked with different connections in the first figure, and a connection is accessible. Client 3 has disconnected in the second figure, although the connection is still active. When a client completes his task, the connection is released, and that connection is made accessible to other clients.

#### 5.2.4 How do we connect an application to JDBC with spring boot?

Now, we're building an application that interacts with a Mysql database. Here are the instructions for creating and configuring JDBC with Spring Boot below-

Maven Dependencies pom.xml

*Remember below version only supports **com.mysql.jdbc.Driver***

```

<version>2.0.0.RELEASE</version>
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.0.0.RELEASE</version>
 <relativePath/>
 </parent>
 <groupId>JBK_Spring_JdbcTemplate</groupId>
 <artifactId>JBK_Spring_JdbcTemplate</artifactId>

```

```

<version>0.0.1-SNAPSHOT</version>
<name>JBK_JDBC_Template</name>
<description>JDBC Template Example for Spring Boot</description>
<properties>
 <java.version>1.8</java.version>
</properties>
<dependencies>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-jdbc</artifactId>
 </dependency>
 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <scope>runtime</scope>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-test</artifactId>
 <scope>test</scope>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-devtools</artifactId>
 </dependency>
</dependencies>
<build>
 <plugins>
 <plugin>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-maven-plugin</artifactId>
 </plugin>
 </plugins>
</build>
</project>

```

## Employee Table

```
CREATE TABLE `jbkdb`.`employee1` (`eid` int(10), `ename` varchar(45),PRIMARY KEY (`eid`));
```

## Pojo class for Employee Database - *Employee.java*

```
public class Employee {
 public Employee(int id, String name) {
 super();
 this.id = id;
 this.name = name;
 }
 public Employee() {

 }
 private int id;
 private String name;

 public int getId() {
 return id;
 }
 public void setId(int id) {
 this.id = id;
 }
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 @Override
 public String toString() {
 return "Employee [id=" + id + ", name=" + name + "]";
 }
}
```

## Mapper class for Employee Table

This class needs to be created for mapping employee pojo and column of table. Generally we can say if I have 10 tables then like this we need to have 10 classes for mapping.

```
package com.javabykiran;
import java.sql.ResultSet;
```

```

import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
public class EmployeeMapper implements RowMapper<Employee> {
 @Override
 public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
 Employee employee = new Employee();
 employee.setId(Integer.parseInt(rs.getString("eid")));
 employee.setName(rs.getString("ename"));
 return employee;
 }
}

```

### Dao class for Database Operation

1. In this class we must autowire JdbcTemplate.
2. This class must be qualified bean so annotate class with repository annotation.

```

package com.javabykiran;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
@Repository
public class EmployeeDao {
 @Autowired
 private JdbcTemplate jdbcTemplate;

 public int saveEmployee(Employee e) {
 String query = "insert into employee
values('" + e.getId() + "','" + e.getName() + "')";
 return jdbcTemplate.update(query);
 }

 public int updateEmployee(Employee e) {
 String query = "update employee
set name='" + e.getName() + "' where eid='" + e.getId() + "'";
 return jdbcTemplate.update(query);
 }

 public int deleteEmployee(Employee e) {
 String query = "delete from employee
where eid='" + e.getId() + "'";

```

```

 return jdbcTemplate.update(query);
 }

 public List<Employee> getEmployee(int id) {
 String query = "select * from employee
 where eid=" + id + " ";
 return jdbcTemplate.query(query, new EmployeeMapper());
 }

 public Employee getEmployeeById(int id) {
 String query = "select * from employee
 where eid=" + id + " ";
 return jdbcTemplate.queryForObject(query, new EmployeeMapper());
 }

 public void employeeOperation(Employee e) {
 saveEmployee(e);
 updateEmployee(e);
 deleteEmployee(e);
 }
}

```

In above program we can see:

1. DML operations can be performed by using update method.
2. Select operation are achieved by query and queryForObject method.
3. Observe in these all scenarios we have control over queries that's beauty of JdbcTemplate.
4. In hibernate most of the time we not get control.
5. In pure jdbc lot of code needs to be written for connection.

#### Application Properties file for DB Details

```

spring.datasource.url=jdbc:mysql://localhost/jbkdb
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

```

Testing Application - Spring JdbcTemplate

Write test cases class for testing. We will be using JUnit for this purpose.

```

package com.javabykiran;
import java.util.List;
import org.junit.Test;
import org.junit.runner.RunWith;

```

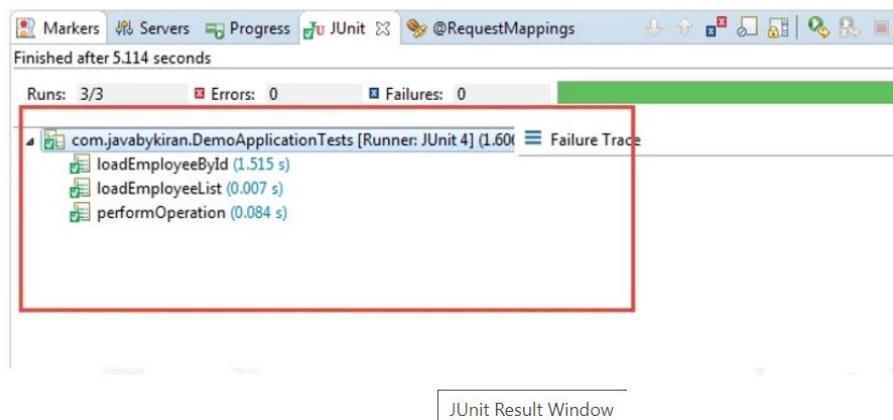
```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
@RunWith(SpringRunner.class)
@SpringBootTest
public class DemoApplicationTests {
 @Autowired
 EmployeeDao employeeDao;

 @Test
 public void contextLoads() {
 employeeDao.employeeOperation(new Employee(19,"kiran19"));
 }
 @Test
 public void loadEmployeeList() {
 List<Employee> listEmp= employeeDao.getEmployee(13);
 System.out.println(listEmp.size());
 }
 @Test
 public void loadEmployeeById() {
 Employee emp= employeeDao.getEmployeeById(13);
 System.out.println(emp);
 }
}

```

We can see result in JUnit window as below.



## Advantages of spring boot JDBC over Spring JDBC

<b>Spring Boot JDBC</b>	<b>Spring JDBC</b>
<b>Spring-boot-starter-JDBC</b> dependency is there, which is required.	Multiple dependencies like <b>spring-JDBC</b> and <b>spring-context</b> need to be configured in Spring JDBC.
Datasource bean is automatically configured if not maintained explicitly, and we can set property called <b>spring.datasource.initialize to false</b> if we don't want to use beans.	Creating a database bean in spring JDBC is necessary either by using <b>XML</b> or <b>javaconfig</b> .
Spring Boot automatically registers beans, so we do not need to register template beans.	Some template beans have to be registered, such as <b>PlatformTransactionManager</b> , <b>JDBCTemplate</b> , <b>NamedParameterJdbcTemplate</b> .
Any DB initialization scripts that are placed in the.sql file are run automatically.	If any database initialization scripts, such as table dropping or creation are created in the SQL file, this information must be explicitly included in the settings.

## 5.2.5 What is the Richardson Maturity Model?

Richardson maturity model is a popular model used to rank your API based on the checks correlated to REST. The more your API fulfills the checks and constraints; the more RESTful your API is for development and deployment. This special model has four stages, which are called levels, and the levels range from 0 to 3. If your API is in Level 0, it means your API is not at all RESTful, and on the other hand of your API is on Level 3, then it can be said that your API is fully RESTful and completing all the constraints of RESTfulness. Let us see what each level holds inexact:

### Level 0

- In this stage, your API will employ only a single URI for exposing the entire API.
- Moreover, usually, it employs only one a single HTTP method for fetching the resources. For example, if you take an API of employee registration, you can see that each of the resources associated with your API has its individual URI. The employee resource has a separate URI; the designation resource has its own as well as the employee-registration resource holds a separate URI.

### Level 1

- The second level possesses its own URI for all of its resources. Moreover, level 1 employs several URIs, which leads to a specific resource, and each URI acts as an entry point. The only POST method is used for the operation.

### Level 2

- This level suggests that your API must employ the properties of the protocol as greatly as feasible. Other than using a particular POST method for performing different operations, developers must implement the GET method also to request a resource. Also, for deleting a resource, the DELETE method needs to be implemented.

### **Level 3**

- According to this level, your API must implement the concept of HATEOAS. Your response must include a logical link(s) for the resources that your API is having. Let's consider the scenario of a case where the client demands the employee's information in which the employee's name is Alex; then the response should possess a logical link containing the URI of all employees that are registered to that company by that name.

## **5.2.6 RESTful Web Services Best Practices:**

### **What Is Maven?**

- Maven is a build and project management tool that is generally used in frameworks built in Java. It is developed by Apache Software Foundation. Maven, a word from the Yiddish language, means 'gatherer of knowledge'. It was introduced to make the process of triggering build in the Jakarta Turbine Project.
- Maven is controlled by the Project Object Model (pom) file. While working with frameworks built-in in Java, we often have to deal with a number of dependencies.
- Maven ensures that project JARs and libraries are downloaded automatically. Only the information relating to the versions of the software and the type of dependencies need to be described in the pom .xml file.
- Maven can take care of projects in Ruby, C#, and other languages. It takes up the task of building projects, their dependencies, and documentation.
- ANT, another tool developed by Apache Software Foundation, is also used for the building and deployment of projects. But Maven is more advanced than ANT. Like ANT, Maven has made the process of building simple. Thus, in short, Maven has made the life of developers easy.

### **Why Use Maven??**

Maven performs the below activities:

- Repository to get the dependencies.
- Having a similar folder structure across the organization.
- Integration with Continuous Integration tools like Jenkins.
- Plugins for test execution.
- It provides information on how the software/ project is being developed.
- The build process is made simpler and consistent.
- Provides guidelines for the best practices to be followed in the project.
- Enhances project performance.
- Easy to move to new attributes of Maven.
- Integration with version control tools like Git.

Maven takes care of processes like releases, distribution, reporting, builds, documentation, and SCMs. Maven connects to the Maven Central repository and loads them locally. Some of the IDEs that support project development with Maven are NetBeans, Eclipse, IntelliJ, and so on.

Maven should be used in our projects in the following scenarios:

- If the project requires a large number of dependencies.
- If the version of the dependencies needs frequent up-gradation.
- If the project needs to have quick documentation, compiling, and packaging of the source code into JAR or ZIP files.

## Operational Steps Of Maven

1. First Maven goes through the POM .xml file.
2. The dependencies are loaded into the local repository.
3. Goes through the built-in life cycles of Maven as shown below:
  - Default: Takes care of the deployment of the project.
  - Clean: Removes any errors, thereby cleaning the project, and removing the artifact produced from the previous process of the build.
  - Site: Takes care of the documentation of the project.
4. Each built-in cycles have several phases. For example, default has twenty-three phases while clean and site have three and four phases respectively.
5. Each Maven cycle goes through several stages where a particular stage has a specific objective.

### Important phases of Maven listed below:

- ✓ Validate Verifies if all the prerequisite data for the build to complete are available.
- ✓ Compile: Source code is compiled.
- ✓ Test- compile: Test source code is compiled.
- ✓ Test: Unit test cases are executed.
- ✓ Package: Source code is compiled and packaged into JAR or ZIP files.
- ✓ Integration- test: The package is deployed and if there are any issues, integration test cases are executed.
- ✓ Install-Package: It is installed in the local repository.
- ✓ Deploy: A copy of the package is made available from the remote repository.

These phases have to be executed in order. Also, if the deploy phase i.e. the end phase of the Maven cycle has to be executed then all the prior phases to that cycle have to be completed successfully.

From the command prompt, the phases are run in the following way:

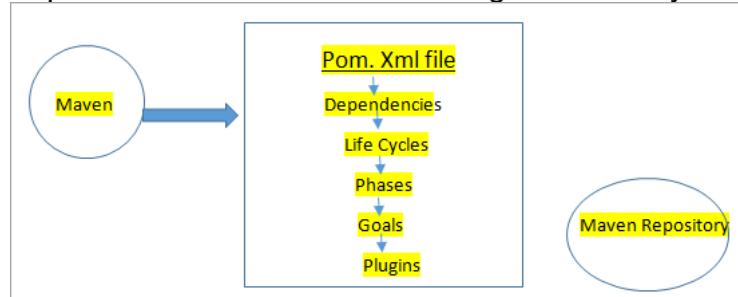
mvn <name of phase>, for example, mvn validate

6. A group of Maven goals makes up a phase. Just like phases of Maven, each goal has to be run in a specific order. A goal has the following syntax:

Here, we have discussed some of the phases along with the goals tied to them:

compiler: compile ( used in the phase of compilation )  
compiler: test ( used in the phase of test compilation )  
surefire: test ( used in the phase of testing )  
install: install ( used in the phase of installation )  
jar: war ( used in the phase of packaging )  
war: war ( used in the phase of packaging )

The operational steps of how Maven works are diagrammatically shown below:



Reference: <https://www.softwaretestinghelp.com/maven-tutorial/>

## Terminologies In Maven

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a Maven project named 'ExcelDriven'. The structure includes source code folders like 'src/main/java' and 'src/main/resources', test code folders like 'src/test/java' and 'src/test/resources', and dependency jars like 'JRE System Library [J2SE]'. The 'pom.xml' file is selected in the Project Explorer. On the right, the Editor view shows the XML content of the 'pom.xml' file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0</modelVersion>
<groupId>Framework</groupId>
<artifactId>ExcelDriven</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
 <!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml/4.1.1 -->
 <dependency>
 <groupId>org.apache.poi</groupId>
 <artifactId>poi-ooxml</artifactId>
 <version>4.1.1</version>
 </dependency>
<!-- https://mvnrepository.com/artifact/org.an -->
```

- POM: It stands for Project Object Model. It is an XML file that has information about the project, the dependencies present in the project, the directory of the source file, plugin information, and so on.
- These are the necessary data for Maven to completely build the project. Maven reads the pom file to get all this information.
- Maven project in Eclipse IDE with a code snippet from the POM XML file is shown below.
  - ✓ GroupId: Recognizes our project uniquely from all the projects. GroupId is a part of the pom file. It is often said as an identity for the group of projects.
  - ✓ ArtifactId: A jar file that is deployed to the Maven repository. ArtifactId is a part of the pom file. It is often said to be the identity and name of our project.
  - ✓ Version: Specifies the version of the jar of the project. Version is also a part of the pom file.
  - ✓ As depicted in the image above, we can see that <groupId> , <artifactId> and <version> tags form the part of the dependencies defined for the project.
  - ✓ Maven Central Repository: This is the placeholder where jars, libraries, plugins, and configuration data required by Maven for building the project are present.

## Maven Repository

Maven Repository can be of three types:

- Local Repository

- Remote Repository
- Central Repository

Once Maven reads the dependencies from the POM file, it first searches them in the local repository, then to the central, and finally to the remote repository. If the dependencies are not found in any of the three repositories, then the user is notified with an error and the process is stopped.

### 1. Maven Local Repository

- The local repository is located in our local system – mostly in .m2 (C:/Users /superdev /.m2) directory which shows its presence once Maven is installed in our system and we have been able to successfully execute a Maven command.
- It is also possible to modify this location in settings.xml (MAVEN\_HOME /conf /settings.xml) with the help of the localRepository tag.
- Below xml code snippet, that shows how to change the locations of the local repository:

```
<settings>
<localRepository>C: \Maven \m2</localRepository>
</settings>
```

### 2. Maven Central Repository

- Central repository is developed by the Apache Maven group and is hosted on the web. This is considered as the central repository and it has all the common libraries. Like a local repository, we can also modify the location where they are to be downloaded by default by changing the setting.xml.

### 3. Maven Remote Repository

- A remote repository is also hosted on the web. In some scenarios, a company can develop its own remote repository and perform deployments on its private projects. These will be owned by that specific company and can be operated only inside it.
- The remote repository has similar working patterns to a central repository. Whenever any dependencies or configurations are required from these repositories, they shall first be downloaded into our local and then used.
- A sample xml code for the remote repository with the id and url is shown below.

```
<repositories><repository>S
<id>com.src.repository</id>
<url>http://maven.comp.com/maven2/</url>
</repository></repositories>
```

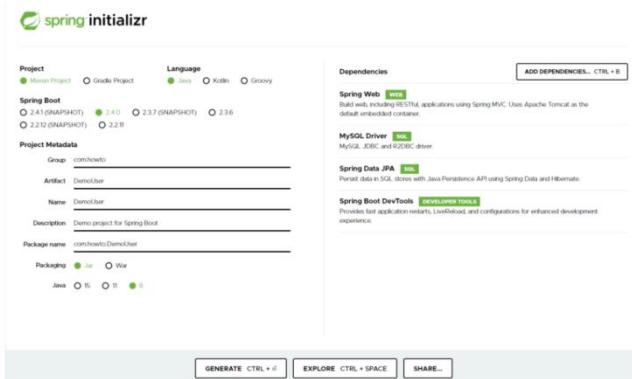
[How to Create a REST API With Spring Boot](#)

**Follow the below-mentioned steps to build a Spring Boot REST API using Java.**

- Step 1: Initializing a Spring Boot Project
- Step 2: Connecting Spring Boot to the Database
- Step 3: Creating a User Model
- Step 4: Creating Repository Classes
- Step 5: Creating a Controller
- Step 6: Compile, Build and Run
- Step 7: Testing the Spring Boot REST APIs

## Step 1: Initializing a Spring Boot Project

- To start with **Spring Boot REST API**, you first need to initialize the Spring Boot Project. You can easily initialize a new Spring Boot Project with Spring Initializr.
- From your Web Browser, go to [start.spring.io](https://start.spring.io). Choose **Maven** as your Build Tool and Language as **Java**. Select the specific version of Spring Boot you want to go ahead with.



Reference: <https://hevodata.com/learn/spring-boot-rest-api/>

You can go ahead and add a few dependencies to be used in this project.

- **Spring Data JPA** – Java Persistence API and Hibernate.
- **Spring Web** – To include Spring MVC and embed Tomcat into your project.
- **Spring Boot DevTools** – Development Tools.
- **MySQL Driver** – JDBC Driver (any DB you want to use).

Once you're done with the configuration, click on “**Generate**”. A ZIP file will then be downloaded. Once the download is finished, you can now import your project files as a Maven Project into your IDE (such as **Eclipse**) or a text editor of choice.

## Step 2: Connecting Spring Boot to the Database

- Next, you need to set up the Database, and you can do it easily with Spring Data JPA.
- Add some elementary information in your **application.properties** file to set up the connection to your preferred Database. Add your JDBC connection URL, provide a username and password for authentication, and set the **ddl-auto** property to **update**.
- Hibernate has different dialects for different Databases. Hibernate automatically sets the dialect for different Databases, but it's a good practice to specify it explicitly.

```
spring.datasource.url = jdbc:mysql://localhost:3306/user
spring.datasource.username = user
spring.datasource.password = user
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect
= org.hibernate.dialect.MySQL5Dialect
```

=

### Step 3: Creating a User Model

- The next step is to create a **Domain Model**. They are also called **Entities** and are annotated by **@Entity**.
- Create a simple User entity by annotating the class with **@Entity**. Use **@Table** annotation to specify the name for your Table. **@Id** annotation is used to annotate a field as the id of an entity. Further, set it as a **@GeneratedValue** and set the **GenerationType** to **AUTO**.

```
@Entity
@Table(name = "user")
public class User {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private long id;
 private String name;
}
```

Now, this class (entity) is registered with Hibernate.

### Step 4: Creating Repository Classes

- To perform CRUD (Create, Read, Update, and Delete) operations on the **User** entities, you'll need to have a **UserRepository**. To do this, you'll have to use the **CrudRepository** extension and annotate the interface with **@Repository**.

```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {}
```

### Step 5: Creating a Controller

You've now reached the Business Layer, where you can implement the actual business logic of processing information.

**@RestController** is a combination of **@Controller** and **@ResponseBody**. Create a **UserController** as shown below.

```
@RestController
@RequestMapping("/api/user")
public class UserController {
 @Autowired
 private UserRepository userRepository;
 @GetMapping
 public List<User> findAllUsers() {
 // Implement
 }
 @GetMapping("/{id}")
 public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
 // Implement
 }
 @PostMapping
 public User saveUser(@Validated @RequestBody User user) {
 // Implement
 }
```

```
}
```

You've to **@Autowired** your **UserRepository** for dependency injection. To specify the type of HTTP requests accepted, use the **@GetMapping** and **@PostMapping** annotations.

Let's implement the **findAll()** endpoint. It calls the **userRepository** to **findAll()** users and returns the desired response.

```
@GetMapping
public List<User> findAllUsers() {
 return userRepository.findAll();
}
```

To get each user by their **id**, let's implement another endpoint.

```
@GetMapping("/{id}")
public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
 Optional<User> user = userRepository.findById(id);

 if(user.isPresent()) {
 return ResponseEntity.ok().body(user.get());
 } else {
 return ResponseEntity.notFound().build();
 }
}
```

If the **user.isPresent()**, a **200 OK** HTTP response is returned, else, a **ResponseEntity.notFound()** is returned.

Now, let's create an endpoint to save users. The **save()** method saves a new user if it isn't already existing, else it throws an exception.

```
@PostMapping
public User saveUser(@Validated @RequestBody User user) {
 return userRepository.save(user);
}
```

The **@Validated** annotation is used to enforce basic validity for the data provided about the user. The **@RequestBody** annotation is used to map the body of the **POST** request sent to the endpoint to the **User** instance you'd like to save.

### Step 6: Compile, Build and Run

You can change the port of Spring Boot from your **application.properties** file.

```
server.port = 9090
```

8080 is the default port that Spring Boot runs in.

It's time to run the Spring Boot REST API you've created. To run the application, directly execute **./mvnw spring-boot:run** on the command line from your base project folder where **pom.xml** is located.

If your application has successfully run, you'll be able to see these audit logs at the end of your command line.

```
2020-11-05 13:27:05.073 INFO 21796 --- [restartedMain]
o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port
35729
2020-11-05 13:27:05.108 INFO 21796 --- [restartedMain]
o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s):
8080 (http) with context path "
```

```
2020-11-05 13:27:05.121 INFO 21796 --- [restartedMain]
com.howto.DemoUser.DemoUserApplication : Started DemoUserApplication
in 1.765 seconds (JVM running for 2.236)
```

### Step 7: Testing the Spring Boot REST APIs

Your Spring Boot REST API is now up and running on <http://localhost:8080/>.

Use your browser, **curl**, or **Postman** to test the endpoints.

To send an HTTP GET request, go to <http://localhost:8080/api/user> from your browser and it will display a JSON response as shown.

```
[
 {
 "id": 1,
 "name": "John"
 },
 {
 "id": 2,
 "name": "Jane"
 },
 {
 "id": 3,
 "name": "Juan"
 }
]
```

Now, send an HTTP POST request to add specific users to your Database.

```
$ curl --location --request POST 'http://localhost:8080/api/user'
--header 'Content-Type: application/json'
--data-raw '{ "id": 4, "name": "Jason" }'
```

The API will return a 200 OK HTTP response with the response body of the persisted user.

```
{
 "id": 4,
 "name": "Jason"
}
```

That's it; you have now successfully tested your Spring Boot REST API.

## Chapter 6: Reviewing of UI/UX Principles

### Learning Outcomes:

- Understand the fundamental concepts of UI/UX
- Gain knowledge of UI/UX Designing
- Understand the difference between UI/UX Design and Web Design

## 6.1 Introduction to UI/UX

User interface (UI) and user experience (UX) are two words that you might hear mentioned frequently in tech circles (and sometimes interchangeably). But what do the terms actually mean, and what does it mean to be a UX or UI designer? UI refers to the screens, buttons, toggles, icons, and other visual elements that you interact with when using a website, app, or other electronic device. UX refers to the entire interaction you have with a product, including how you feel about the interaction. While UI can certainly have an impact on UX, the two are distinct, as are the roles that designers play. In this chapter, we'll take a closer look at how the roles of UX designer and UI designer overlap and differ, and how to know which you should pursue. Finally, we'll discuss options for getting started, even if you don't have a degree or previous experience.

### Difference between UI and UX

Developing a product that people love often requires both good UI and good UX. For example, you could have a banking app that looks great and has intuitive navigation (UI). But if the app loads slowly or makes you click through numerous screens to transfer money (UX), it doesn't matter how good it looks. You're probably not going to want to use it. On the other hand, a website could be loaded with unique, helpful content organized in a logical and intuitive way. But if it looks dated or you can't easily figure out how to move between screens or scroll through options, you're likely to click away from the site.

UX designer	UI designer
 Interaction designer	 Visual designer
 Charts the user pathway	 Chooses color and typography
 Plans information architecture	 Plans visual aesthetic
 Expert in wireframes, prototypes, and research	 Expert in mockups, graphics, and layouts

Reference: [UI vs. UX Design: What's the Difference? | Coursera](#)

### Tasks and responsibilities: What do they do?

Both UI and UX designers play key roles in the product development lifecycle. Let's take a closer look at each.

**UX designers** focus their work on the experience a user has with a product. The goal is to make products that are functional, accessible, and enjoyable to use. While the term UX often applies to digital products, it can also be applied to non-digital products and services (like a coffee pot or a transportation system). Common tasks for a UX designer might include:

- Conducting user research to identify any goals, needs, behaviors, and pain points involved with a product interaction
- Developing user personas based on target customers
- Creating user journey maps to analyze how a customer interacts with a product
- Building wireframes and prototypes to hone in on what the final product will look like
- Performing user testing to validate design decisions and identify problems
- Collaborating with stakeholders, UI designers, and developers

**UI designers** create the graphical portions of mobile apps, websites, and devices the elements that a user directly interacts with. Unlike UX, which can apply to just about any product or service, the term UI applies exclusively to digital products. A UI designer seeks to make apps and websites both visually appealing and easy to navigate. Common tasks of a UI designer include:

- Organizing page layouts
- Choosing color palettes and fonts
- Designing interactive elements, such as scrollers, buttons, toggles, drop-down menus, and text fields
- Making high-fidelity wireframes and layouts to show what the final design will look like
- Working closely with developers to convert designs into a working product

## 6.2 Five-key Principles of UX Design

While there are numerous principles that contribute to effective UX design, here are five key principles that can greatly influence the overall user experience:

### 1. Hierarchy:

Establishing a clear visual hierarchy helps users understand the relative importance and relationships between different elements within a design. By utilizing techniques like size, color, contrast, and spacing, designers can guide users' attention and focus on the most critical information or actions. This principle ensures that users can easily navigate and comprehend the content or interface.

### 2. Consistency:

Consistency is crucial in UX design as it creates familiarity and predictability for users. By maintaining consistent visual elements, such as typography, colors, icons, and layout, throughout the entire product or across different screens, users can easily understand and recognize patterns. Consistency enhances learnability, reduces cognitive load, and makes the overall experience more cohesive and intuitive.

### **3. Confirmation:**

Providing feedback and confirmation to users when they perform actions is essential for creating a sense of control and understanding. Designers should implement clear and timely feedback for user interactions, such as button clicks, form submissions, or error messages. Confirming user actions through visual cues or messages reassures users that their actions have been recognized and understood by the system, preventing confusion or frustration.

### **4. User Control:**

Empowering users with a sense of control and autonomy over their interactions is a fundamental principle of UX design. Users should be able to easily navigate, manipulate, and customize the interface according to their preferences. Providing options for undo/redo, adjustable settings, intuitive navigation, and clear signifiers for interactive elements (e.g., buttons, links) allows users to feel in control and enhances their overall satisfaction with the product.

### **5. Accessibility:**

Accessibility ensures that digital products are usable by individuals with disabilities and provide an inclusive experience for all users. Designers should consider factors like color contrast, font sizes, alternative text for images, keyboard navigation, and screen reader compatibility. Incorporating accessibility guidelines and standards into the design process enables a wider range of users to access and interact with the product effectively.

These principles serve as a foundation for designing user-centered experiences that prioritize clarity, ease of use, and inclusivity. By incorporating these principles into the design process, UX designers can create intuitive and engaging products that meet the needs and expectations of their users.

## **6.3 Five Key UI Design Principles**

Here are five key UI design principles that contribute to creating effective and user-friendly interfaces:

### **1. Clarity:**

Clarity is a fundamental UI design principle that emphasizes the importance of clear and understandable interfaces. It involves presenting information, content, and functionality in a way that is easily comprehensible to users.

Here's a concise explanation of the Clarity principle:

**Clarity in UI Design:** The Clarity principle states that UI designs should be clear, straightforward, and easily understandable to users. It focuses on minimizing confusion and cognitive load by presenting information and interactions in a concise and unambiguous manner.

Key aspects of the Clarity principle include:

1. Simplified Visuals: Use clean and minimalist design approaches, eliminating unnecessary visual clutter and distractions. Prioritize clarity by removing irrelevant elements and ensuring a balanced use of white space.
2. Clear Hierarchy: Establish a clear visual hierarchy to guide users' attention and indicate the relative importance of different elements. Use size, color, typography, and spacing to differentiate between primary, secondary, and tertiary elements.
3. Readability: Prioritize legibility by choosing appropriate fonts, font sizes, and contrast levels. Ensure that text is easily readable, both in terms of content and formatting. Use headings, subheadings, bullet points, and proper spacing to enhance readability.
4. Intuitive Navigation: Design navigation structures that are intuitive and easy to understand. Use recognizable icons, clear labels, and consistent placement to guide users and help them find what they need without confusion.
5. Feedback and Clarity of Interactions: Provide clear and immediate feedback to users when they interact with elements or perform actions. Use visual cues, animations, or notifications to indicate the system's response to user input.
6. Error Prevention and Handling: Design error messages and handling in a clear and informative way. Clearly communicate any errors or issues that occur, offering guidance on how to resolve them effectively.

By adhering to the Clarity principle, UI designers create interfaces that are easily understood and navigated by users. Clarity reduces cognitive load, enhances user satisfaction, and ensures a positive user experience.

## 2. Consistency in Structure:

While designing the user interface, it is crucial to be mindful of the interactions that take place between the screen you are designing for and human cognition. Consistency in structure serves to make things more approachable to your users, without feeling overwhelming or messy

Here Are A Few Things You Can Do To Ensure Consistency:

- Incorporate impactful visual hierarchy, with the most vital things bold and big
- Throughout the app or website, strive to use a consistent color theme.
- Introduce a system of visual order, such as aligning everything along the grid
- Navigation should be kept constant across all screens.

- It's important to leverage the same fonts and types in your elements, especially when contending with form elements.
- Same elements should be re-purposed for different situations. For instance, the same sample notification can be color coded to serve in myriad situations.
- Keep all of your borders on images, galleries, selects, inputs, and buttons consistent with each other.
- Background images should not be changed frequently from page view to page view, or all hero images should at least be relatable to each other so that consistency across pages is maintained.
- Commonly used UI elements, such as radio buttons, scrollbars, and icons, are typically consistent graphic elements and have representations that are known and understood by users widely. For instance, we use radio buttons when users are afforded only one option, while checkboxes are employed to offer multiple choices to a user.
- When deciding on a layout, it's prudent to factor in well-established convention. Humans have a strong memory when it comes to where things are visually located on the screen. This characteristic should be capitalized on by placing various graphical elements on commonly used locations, such as having the exit icon on the top right, search field on the top right, and logo on the top left.
- Make sure that your website incorporates all the functionalities and features that users are expecting to find on your website. For instance, a music sharing site is expected to have an embedded media player, while a website for airlines should have a viable ticket-booking system.

Leveraging existing user mental models and familiar design patterns can enhance the usability of an interface. UI elements and interactions that align with established conventions and user expectations make it easier for users to understand and interact with the interface. Consistent use of common icons, menus, and interactions across platforms or similar products creates a sense of familiarity, reducing the learning curve and promoting a smoother user experience.

### **3. User Control:**

It's hard to say which is the most important UI design principle, but you'll find this one at the top of most lists.

**What use is a vehicle if the controls don't feel within reach and instantly intuitive?** Good UI design should give users an immediate sense of comfort and control. This will allow them to learn and master a product quickly and easily.

By that token, user learning will be slowed when they grapple with a tricky-to-control product.

There is, however, a balance to be struck. Too much freedom of control, or unnecessary complexity, and the user will become overwhelmed and exhausted. Decisions demand brainpower, after all. So how much control is enough?

- **Navigation:**
  - An easily navigated interface is a crucial element of control within good UI design. Navigation should be made so clear and obvious that it becomes an unconscious, enjoyable experience for the user, rather than a nerve-racking one.
  - This can be achieved by providing the user with information and context regarding where they are, where they came from, and where they are going next.
  - Visual cues, prompts, and feedback will put the user in their navigation comfort zone. Bombard them with too many features and you may find navigation suffers as a result.
  
- **Signposts:**
  - User murmurs of “Where am I?”, or “How did I get here?” are signs of a UI design fail. Clear signposting can ensure our user friends are never left wondering about their orientation within the interface.
  
- **Feedback:**
  - While signposts tell users where they are, information feedback keeps them abreast of interactions that are taking place within the product.
  - These can come in the form of color-shifting buttons, audio prompts, or readable UI copy appearing in response to user actions.
  - For example, a frequent and small action, such as hovering over a button, could result in a subtle shift in the color of that element.
  - A more major, infrequent action, such as changing your email password or transferring money from one account to another, could trigger text prompts and warnings.
  
- **Undo:**
  - Another way of minimizing feelings of jeopardy in UI is to incorporate a safety net of some kind into the interface.
  - Knowing that any act (or even a long sequence of adjustments) can be undone—and in turn, reinstated—frees the user up to play and explore the interface, without risking a loss of work.
  - This is very common in the realm of graphic design UI, e.g. the Undo and Redo buttons in Photoshop. Simple Back and Forward buttons in online forms provide the same feeling of security.
  
- **Shortcuts:**
  - For more seasoned users, the addition of shortcuts can provide new levels of mastery within a product.
  - This loops back to our point earlier about the balance of control. Shortcuts offer a learning curve for users that want to take control of an interface up a gear, without overwhelming or excluding more novice users.

- While commands such as ‘copy’ and ‘paste’ have entered common parlance, combinations of obscure shortcuts still offer deep levels of control unavailable to casual users.

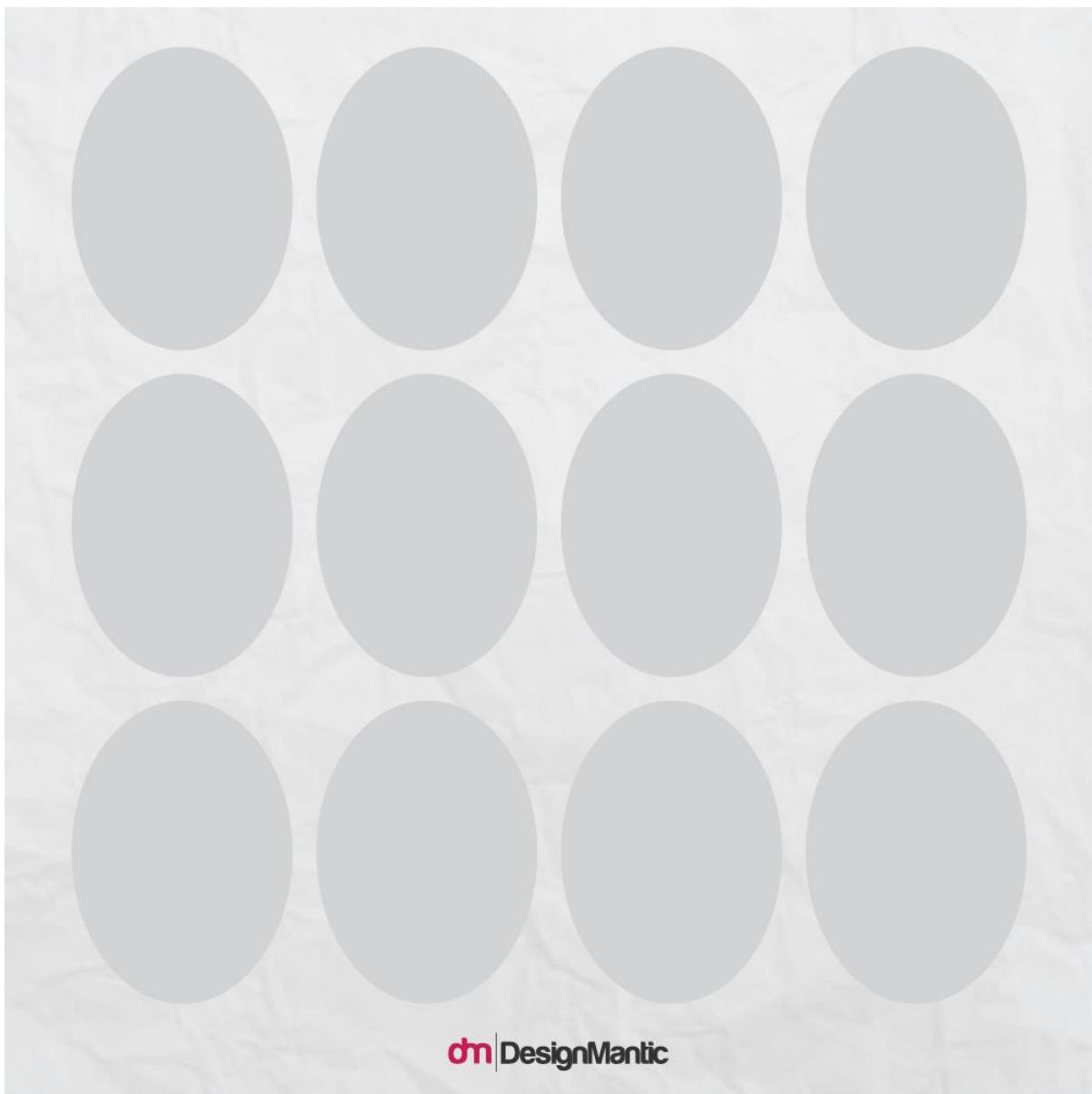
Providing users with a sense of control over their interactions is crucial for a positive UI experience. Users should be able to easily navigate, manipulate, and customize the interface based on their preferences. Offering adjustable settings, allowing users to undo/redo actions, providing clear feedback for their interactions, and avoiding disruptive interruptions empower users and enhance their overall satisfaction.

#### **4. Visual Hierarchy:**

Visual hierarchy is a fundamental characteristic of a great web user interface design and is achieved when elements on a screen are organized in a clear viewing order, allowing users to fish out the most important information before focusing on the less vital one. The primary information, such as the company name, logo, and the navigation menu, should be placed in the center position and rendered in design elements that stand out to make the vital information instantly conspicuous to users.

To grasp the importance of visual hierarchy, look at the image below and tell us which element captured your attention first and guided your eye? The answer is probably not any single element since they all look alike!

Which one did you see at first? Maybe the first circle, or the middle one or all of them. The point is that nothing guided our eyes because all the circles look the same.



On the other hand, look at the image below and think of where your eye was instantly diverted to?



**dm** | DesignMantic

The answer must be the red circle as it stands out the most and is hard to overlook. This is the power of visual hierarchy!

Several factors affect how humans perceive information and how they rank the hierarchy of the content within the website layout. Some factors that have an impact on hierarchy include:

#### Factors Effecting Visual Hierarchy In Web

- **Size:** Compared to smaller elements, larger ones elicit more attention and draw the eye of the users. Notice how the headlines of Newspapers dominate and goad you to find out what the piece is all about.
- **Color:** when compared to dull and drab shades, vibrant and perky hues capture the eye, followed by darker and richer ones. Lighter tints follow in the spectrum as they seem distant and washed out, proceeded by muted or subdued/greyscale colors.

- **Contrast:** If you want to draw the eye of the user to an element on your website, dramatically contrasting colors fare much better than slightly contrasting hues. Contrast helps to highlight the most important parts of your site. Everything else is relative.
- **Alignment:** Alignment helps to lend a semblance of order to design elements. For instance, placing a sidebar column next to content creates a priority for the user.
- **Repetition:** Repetition in style gives a feeling that parts of content are related. For instance, since most of the content is presented in the same style on this page, parts that aren't such as boxed text, instantly capture attention.
- **Proximity:** According to Gestalt, The closer the design elements are placed together, the more related they appear to users.
- **Whitespace:** The space around your content can be used to draw the eye to specific parts of the content. For instance, white spaces break up content into easily digestible parts and allow the eye to zero in on various parts of the page.
- **Style and Texture:** The use of styles and textures can help to prioritize content for users. They can be used to set the tone of the design like fonts.

Establishing a clear visual hierarchy helps users prioritize and understand the relative importance of different elements within the interface. Visual cues like size, color, contrast, and spacing can guide users' attention and convey information hierarchy effectively. Important elements or actions should stand out, while less critical ones should be appropriately de-emphasized. A well-defined hierarchy improves usability and prevents users from feeling overwhelmed.

## 5. Flexibility:

Flexibility is a key principle in UI design that focuses on providing users with adaptable and customizable experiences to meet their individual needs and preferences. Here's a brief explanation of the flexibility principle in UI design:

**Flexibility Principle in UI Design:** The flexibility principle states that UI designs should offer flexibility and adaptability to accommodate diverse user requirements, workflows, and usage contexts. It involves providing options and features that allow users to tailor the interface to their liking and optimize their interactions.

Key aspects of the flexibility principle include:

1. **Customization:** UI designs should allow users to customize elements such as color themes, font sizes, layout preferences, and interface configurations. Customization empowers users to personalize the interface based on their preferences, making it more comfortable and usable for them.
2. **User Preferences:** Incorporate settings and preferences that allow users to adjust the behavior of the interface. This may include options related to notifications, language preferences, privacy settings, or accessibility features. Providing user preferences enhances the user experience by aligning the interface with individual needs and preferences.

3. **Adaptable Layouts:** Design layouts that can adapt to different screen sizes, orientations, and device types. This ensures a consistent and optimized experience across various devices and allows users to interact with the interface seamlessly, regardless of the device they are using.
4. **Contextual Flexibility:** Provide features that adapt to the user's context or usage patterns. This may involve offering different modes or views for different tasks, remembering user preferences or previous interactions, and dynamically adjusting the interface based on user behavior or environment.

By incorporating the flexibility principle, UI designers create interfaces that can be customized, personalized, and adapted to meet the unique needs and preferences of individual users. Flexibility enhances user satisfaction, accommodates diverse usage scenarios, and supports a more engaging and user-centric experience.

Designing interfaces that are flexible and adaptable to different user needs and contexts is essential. Users have varying preferences, screen sizes, input methods, and accessibility requirements. UI designs should accommodate these diverse scenarios and offer responsive layouts, scalable components, adjustable settings, and support for different devices and screen orientations. Flexibility ensures that users can engage with the interface comfortably, regardless of their specific circumstances.

By applying these UI design principles, designers can create interfaces that are visually appealing, intuitive to use, and provide a seamless experience for users.

## 6.4 How Companies Are Engaging Their Users with Real-Time on UX/UI.

Companies are increasingly recognizing the value of real-time engagement in UX/UI design to create dynamic and interactive experiences for their users. Here are some ways companies are engaging their users in real-time:

1. **Real-time Feedback:** Companies are incorporating real-time feedback mechanisms within their interfaces to gather instant user input. This can include live chat features, feedback forms, or interactive elements that allow users to provide feedback or ratings in the moment. Real-time feedback helps companies understand user preferences, pain points, and make improvements to the user experience.
2. **Personalization:** Real-time personalization tailors the user experience based on individual user data and behavior. Companies use real-time data analysis to understand user preferences and dynamically present personalized content, recommendations, or suggestions. This creates a more customized and engaging experience for users.

3. **Real-time Notifications and Alerts:** Companies use real-time notifications and alerts to keep users informed about important updates, events, or actions. These notifications can be delivered through various channels such as push notifications, emails, or in-app messages. Real-time notifications help maintain user engagement and provide timely information.
4. **Live Collaboration:** Some products, especially those in collaboration or communication spaces, offer real-time collaborative features. These allow users to work together simultaneously, seeing each other's changes or contributions in real-time. Real-time collaboration fosters a sense of connectedness and enhances teamwork and productivity.
5. **Interactive and Responsive Interfaces:** Companies are incorporating real-time interactive elements and responsive interfaces that react and respond to user actions in real-time. This can include dynamic visual effects, animations, and real-time data updates. Interactive and responsive interfaces make the user experience more engaging and interactive.
6. **Live Customer Support:** Real-time engagement extends to customer support, where companies offer live chat or real-time support channels. Users can connect with customer support representatives in real-time to get immediate assistance, enhancing customer satisfaction and resolving issues promptly.

By leveraging real-time engagement in UX/UI design, companies create more interactive, personalized, and responsive experiences. Real-time features help companies understand user needs, maintain user engagement, and provide a seamless and dynamic user experience.

## Chapter 7: Cloud Computing for hosting Software Applications

### Learning Outcomes:

- Understand the fundamental concepts of Cloud Computing
- Gain knowledge of Microsoft Azure Portal
- Learn to use Microsoft Azure Services
- Deploy applications on cloud platform

### 7.1 Introduction to Cloud Computing

#### 7.1.1 Introduction to Cloud

Everyone has an opinion on what is cloud computing. It can be the ability to rent a server or a thousand servers and run a geophysical modelling application on the most powerful systems available anywhere. It can be the ability to rent a virtual server, load software on it, turn it on and off at will, or clone it ten times to meet sudden workload demand. It can be

storing and securing immense amounts of data that is accessible only by authorized applications and users.

It can be supported by a cloud provider that sets up a platform that includes the OS, Apache, a MySQL™ database, Perl, Python, and PHP with the ability to scale automatically in response to changing workloads.

Cloud computing can be the ability to use applications on the Internet that store and protect data while providing a service — anything including email, sales force automation and tax preparation. It can be using a storage cloud to hold application, business, and personal data. And it can be the ability to use a handful of Web services to integrate photos, maps, and GPS information to create a mashup in customer Web browsers.

### 7.1.2 Definition

Cloud computing is the delivery of computing services over the Internet. Various computing services and resources such as servers, storage, databases, networking, software, analytics, and intelligence, can be provisioned over the Internet. The ability to dynamically allocate and deallocate the resources allows the user to scale as per need. Pay per use model adds an added advantage. Thus, the end-user can cut the operational cost and simply focus on other business needs. In brief cloud is essentially a bunch of commodity computers networked together in same or different geographical locations, operating together to serve a number of customers with different need and workload on demand basis with the help of virtualization. Cloud services are provided to the cloud users as utility services like water, electricity, telephone using pay-as-you-use business model.

These utility services are generally described as XaaS (X as a Service) where X can be Software or Platform or Infrastructure etc. Cloud users use these services provided by the cloud providers and build their applications in the internet and thus deliver them to their end users.

Cloud is essentially provided by large distributed data centers. These data centers are often organized as grid and the cloud is built on top of the grid services. Cloud users are provided with virtual images of the physical machines in the data centers. This virtualization is one of the key concepts of cloud computing as it essentially builds the abstraction over the physical system.

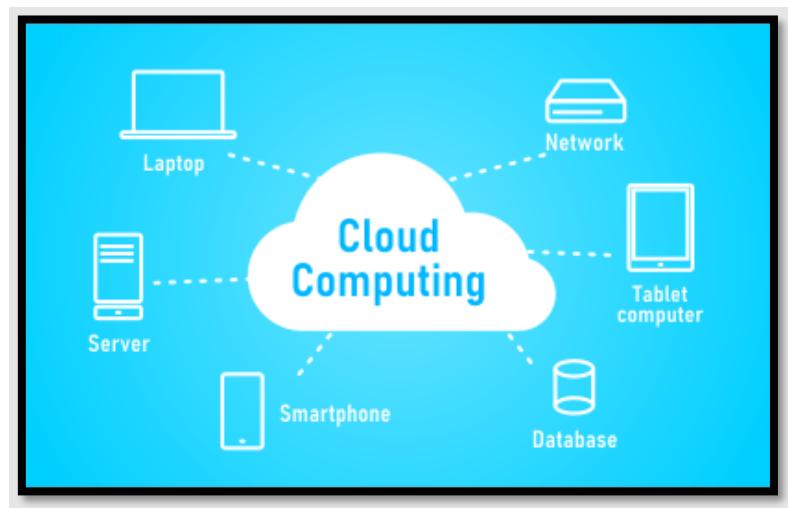


Image: What is Cloud Computing?

Reference: <https://www.thinkitsolutions.com/wp-content/uploads/2019/04/what-is-cloud-computing-with-example-300x187.png>

Many cloud applications are gaining popularity day by day for their availability, reliability, scalability and utility model. These applications made distributed computing easy as the critical aspects are handled by the cloud provider itself.

Cloud computing is growing now-a-days in the interest of technical and business organizations but this can also be beneficial for solving social issues. In the recent time E-Governance is being implemented in developing countries to improve efficiency and effectiveness of governance. This approach can be improved much by using cloud computing instead of traditional ICT. In India, economy is agriculture based and most of the citizens live in rural areas. The standard of living, agricultural productivity etc can be enhanced by utilizing cloud computing in a proper way. Both of these applications of cloud computing have technological as well as social challenges to overcome.

### 7.1.3 Why Cloud Computing?

Let's say your team has a terrific business idea. You plan to develop it technically, showcase it your college and other TechSaksham members. You are all set to go. Then all of a sudden, you hit a setback. You do not have the necessary computing resources. You ask your teammates but no luck there too. You desperately search for buying options but they are expensive. Since they would be used for a very short span of time, it becomes even more infeasible. What would you do?

Here comes the wonderful Cloud Computing. You realize you don't have to buy these resources. You can actually rent out these resources and the best part is that you don't even have to maintain the resources. The provider will do that too. This is why Cloud Computing. You will pay according to the usage.

### 7.1.4 Cloud Computing Basics

Cloud computing is a paradigm of distributed computing to provide the customers on-demand, utility-based computing services. Cloud users can provide more reliable, available and updated services to their clients in turn. Cloud itself consists of physical machines in the data centres of cloud providers. Virtualization is provided on top of these physical machines.

The cloud services are divided into many types like Software as a Service, Platform as a Service or Infrastructure as a Service. These services are available over the Internet in the whole world where the cloud acts as the single point of access for serving all customers. Cloud computing architecture addresses difficulties of large-scale data processing.

### 7.1.5 Types of Cloud

The cloud provides 4 types of deployment models.

**1. Private Cloud** – This type of cloud is maintained within an organization and used solely for their internal purpose. So, the utility model is not a big term in this scenario. Many companies are moving towards this setting and experts consider this is the 1st step for an organization to move into cloud. Security, network bandwidth are not critical issues for private cloud.

**2. Public Cloud** – In this type an organization rents cloud services from cloud provider's on-demand basis. Services provided to the users using utility computing model.

**3. Community Cloud** -- In this type, a specific community of consumer's band together to build a cloud. Similar to a private club cloud. More commonly talked about a decade ago. Not a go-to concept of today.

**4. Hybrid Cloud** – This type of cloud is composed of multiple internal or external cloud. This is the scenario when an organization moves to public cloud computing domain from its internal private cloud.

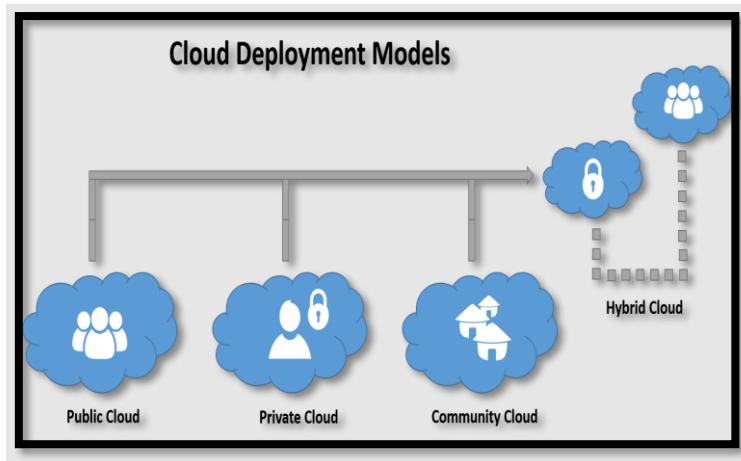


Image: Deployment models of Cloud Computing

Reference: <https://www.uniprint.net/wp-content/uploads/2017/05/Cloud-deployment-structures-diagram.png>

### 7.1.6 Types of Cloud Computing Services

Cloud Computing Services provided by the cloud provider can be classified by the type of the services. These services are typically represented as XaaS where we can replace X by Infrastructure or Platform or Hardware or Software or Desktop or Data etc. There are three main types of services most widely accepted - Software as a Service, Platform as a Service and Infrastructure as a Service. These services provide different levels of abstraction and flexibility to the cloud users. This is shown in the Figure

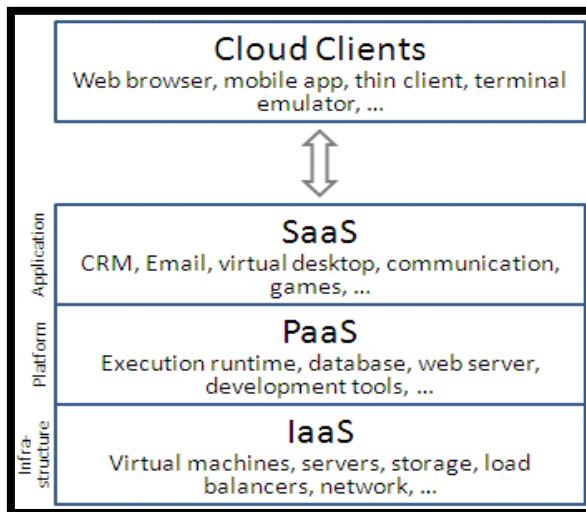


Image: Cloud Service Stack

Reference:

[https://upload.wikimedia.org/wikipedia/commons/3/3c/Cloud\\_computing\\_layers.png](https://upload.wikimedia.org/wikipedia/commons/3/3c/Cloud_computing_layers.png)

We'll now discuss some salient features of some of these models –

#### SaaS (Software as a service)

Delivers a single application through the web browser to thousands of customers using a multitenant architecture. On the customer side, it means no upfront investment in servers or software licensing; on the provider side, with just one application to maintain, cost is low compared to conventional hosting.

For example, Salesforce.com with yearly revenues of over \$300M, offers on-demand Customer Relationship Management software solutions. This application runs on Salesforce.com's own infrastructure and delivered directly to the users over the Internet.

Salesforce does not sell perpetual licenses but it charges a monthly subscription fee starting at \$65/user/month. Google docs is also a very nice example of SaaS where the users can create, edit, delete and share their documents, spreadsheets or presentations whereas Google have the responsibility to maintain the software and hardware. E.g. – Microsoft 365, Zoho Office

#### *PaaS (Platform as a service)*

Delivers development environment as a service. One can build his/her own applications that run on the provider's infrastructure that support transactions, uniform authentication, robust scalability and availability. The applications built using PaaS are offered as SaaS and consumed directly from the end users' web browsers. This gives the ability to integrate or consume third party web-services from other service platforms. E.g. – Azure Function

#### *IaaS (Infrastructure as a Service)*

IaaS service provides the users of the cloud greater flexibility to lower level than other services. It gives even CPU clocks with OS level control to the developers. E.g. – Azure VM and Azure Blob store.

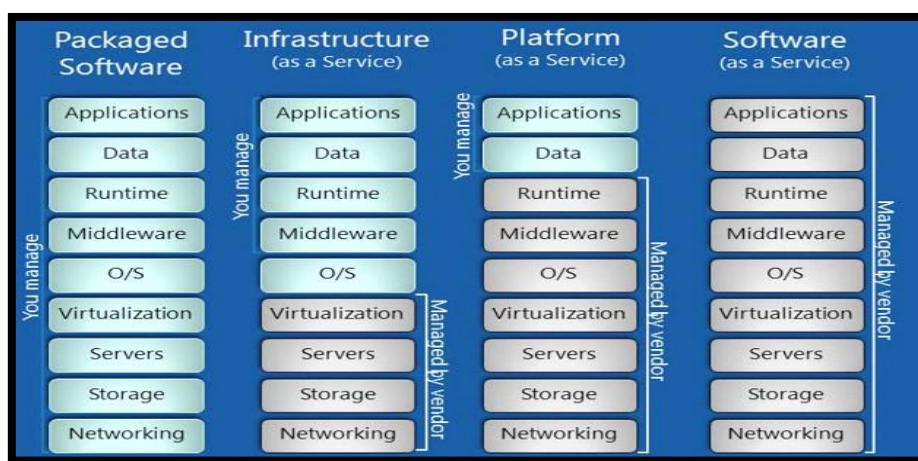


Image: Cloud Service Delivery Models Comparison

Reference: <https://media-exp1.licdn.com/dms/image/C4E12AQGLyzIDZJD5Tw>

### 7.1.7 Advantages of Using Cloud

The advantages for using cloud services can be of technical, architectural, business etc.

#### Cloud Provider's Point of View

- Most of the data centers today are under-utilized. They are mostly 15% utilized. These data centers need spare capacity just to cope with the huge spikes that sometimes get in the server usage. Large companies having those data centres can easily rent those computing power to other organizations and get profit out of it and also make the resources needed for running data centre (like power) utilized properly.
- Companies having large data centers have already deployed the resources and to provide cloud services they would need very little investment and the cost would be incremental.

#### Cloud Users' Point of View

- Cloud users need not to take care about the hardware and software they use and also, they don't have to be worried about maintenance. The users are no longer tied to someone traditional system.
- Virtualization technology gives the illusion to the users that they are having all the resources available.

- Cloud users can use the resources on demand basis and pay as much as they use. So, the users can plan well for reducing their usage to minimize their expenditure.
- Scalability is one of the major advantages to cloud users. Scalability is provided dynamically to the users. Users get as much resources as they need. Thus, this model perfectly fits in the management of rare spikes in the demand

#### 7.1.8 Cloud Architecture

The cloud providers actually have the physical data centres to provide virtualized services to their users through Internet. The cloud providers often provide separation between application and data. This scenario is shown in the image below. The underlying physical machines are generally organized in grids and they are usually geographically distributed. Virtualization plays an important role in the cloud scenario. The data centre hosts provide the physical hardware on which virtual machines resides. User potentially can use any OS supported by the virtual machines used.

Operating systems are designed for specific hardware and software. It results in the lack of portability of operating system and software from one machine to another machine which uses different instruction set architecture. The concept of virtual machine solves this problem by acting as an interface between the hardware and the operating system called as system VMs.

Another category of virtual machine is called process virtual machine which acts as an abstract layer between the operating system and applications. Virtualization can be very roughly said to be as software translating the hardware instructions generated by conventional software to the understandable format for the physical hardware.

Virtualization also includes the mapping of virtual resources like registers and memory to real hardware resources. The underlying platform in virtualization is generally referred to as host and the software that runs in the VM environment is called as the guest. The Figure shows very basics of virtualization.

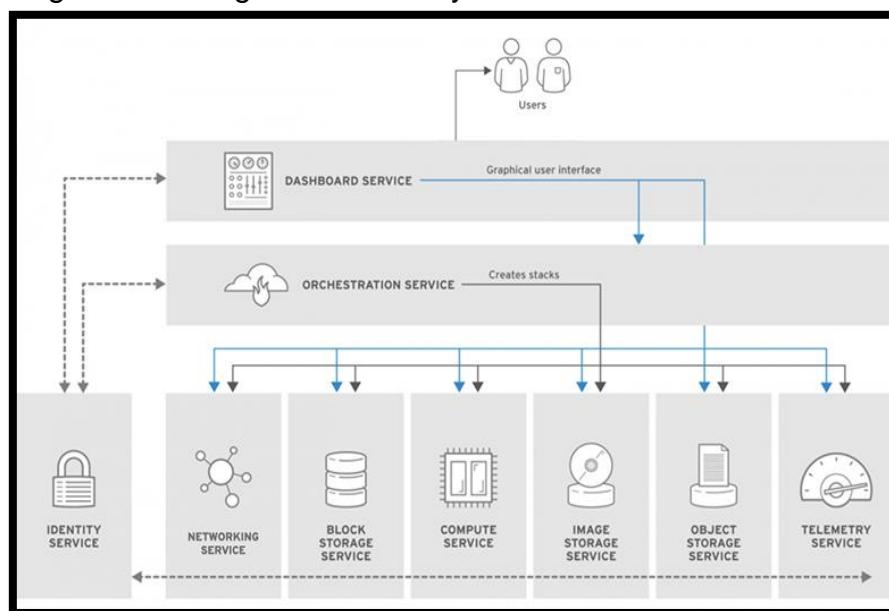


Image: Basics Cloud Computing Architecture

Reference: [https://www.redhat.com/cms/managed-files/styles/wysiwyg\\_full\\_width/s3/Screen%20Shot%202019-06-24%20at%202.27.06%20PM.png?itok=7yByods-](https://www.redhat.com/cms/managed-files/styles/wysiwyg_full_width/s3/Screen%20Shot%202019-06-24%20at%202.27.06%20PM.png?itok=7yByods-)

Here the virtualization layer covers the physical hardware. Operating System accesses physical hardware through virtualization layer. Applications can issue instruction by using OS interface as well as directly using virtualizing layer interface. This design enables the users to use applications not compatible with the operating system. Virtualization enables the migration of the virtual image from one physical machine to another and this feature is useful for cloud as by data locality lots of optimization is possible and also this feature is helpful for taking back up in different locations. This feature also enables the provider to shut down some of the data center physical machines to reduce power consumption.

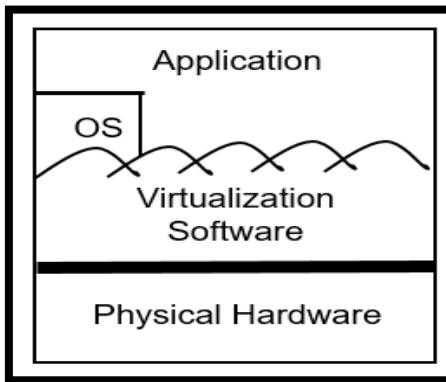


Image: Virtualization Basics

#### 7.1.9 Cloud Computing Application

Today most of the studies in cloud computing is related to commercial benefits. But this idea can also be successfully applied to non-profit organizations and to the social benefit. In the developing countries like India Cloud computing can bring about a revolution in the field of low-cost computing with greater efficiency, availability and reliability.

Recently in these countries e-governance has started to flourish. Experts envisioned that utility-based computing has a great future in e-governance. Cloud computing can also be applied to the development of rural life in India by building information hubs to help the concerned people with greater access to required information and enable them to share their experiences to build new knowledge bases. The major areas of application for cloud computing can be:

- Application Hosting
- Backup and Storage
- Content Delivery
- Websites
- Enterprise IT
- Databases

## 7.2 Cloud Fundamentals

### 7.2.1 Cloud Terminology

- **High Availability** - Accessible whenever you need it

- **Fault Tolerance** - Ability to withstand certain amount of failure and still remain functional
- **Scalability** - Ability to easily grow in size, capacity and/ or scope when required. Growth is usually based on demand.
- **Elasticity** - Ability to grow or scale when required and reduce in size when resources are no longer needed.
- **Cloud bursting** - A configuration which is set up between a private cloud and a public cloud. If 100 percent of the resource capacity in a private cloud is used, then overflow traffic is directed to the public cloud using cloud bursting
- **DevOps** - The union of people, process and technology to enable continuous delivery of value to customers. The practice of DevOps brings development and operations teams together to speed software delivery and make products more secure and reliable.
- **Middleware** - Software that lies between an operating system and the applications running on it. It enables communication and data management for distributed applications, like cloud-based applications, so, for example, the data in one database can be accessed through another database. Examples of middleware are web servers, application servers and content management systems.
- **Serverless Computing** - A computing model in which the cloud provider provisions and manages servers. It enables developers to spend more time building apps and less time managing infrastructure.
- **Virtual Machine** - A computer file (typically called an image) that behaves like an actual computer. Multiple virtual machines can run simultaneously on the same physical computer.
- **Computer Grids** - Groups of networked computers that act together to perform large tasks, such as analyzing huge sets of data and weather modelling. Cloud computing lets you assemble and use vast computer grids for specific time periods and purposes, paying only for your usage and saving the time and expense of purchasing and deploying the necessary resources yourself.
- **Virtualization** - The act of creating a virtual rather than a physical version of a computing environment, including computer hardware, operating system, storage devices and so forth.

### 7.2.2 Characteristics of Cloud Computing

Back in 2011, the National Institute of Standards & Technology, USA drafted some essential characteristics of a cloud model.

They are as follows:

#### **Broad network access**

- Services being available through the Internet (since most cloud computing is public cloud).
- Anyone with a laptop, table, or cell phone should be able to access the computing resources.
- These characteristics rule out an on-premise data center behind a company firewall.
- The network access need not be explicitly Internet. (This integrates IOT/MANET)
- It has also become common for cloud providers to provide a dedicated fiber connection between the customer and provider data center.

- Some vendors also offer their cloud services in “boxes” that are completely cut off from any network. e.g., Nuclear Football (USA) and Cheget (Russia).

### Rapid elasticity

- Computing resources may scale up and down rapidly.
- For example, If a service like a website suddenly has a lot more users because of a flash sale, it needs to be able to scale up the computing power to handle this quickly. Then, when the sale is over, it should scale back down like an elastic band that stretches and retracts when force is no longer applied.
- Not all cloud services come with this automatically. Rather it often needs to be built by the customer.

### Measured Service

- Customers should only pay for what they use to avoid paying for a machine standing idle in a data centre.
- This should be transparent to the customer.
- In practice the unit used to measure can differ greatly. It could be time, storage, processing capacity, number of users, number of instructions, or any other measurable property of relevance to the service.
- The most common are time and number of users.
- It can be very difficult or impossible for a customer to validate the correctness of the metering.

### On-demand self-service

- End users to be able to provision computing resources by themselves.
- Earlier, every request was provisioned by the vendor. Thus, time-consuming and not cost-savvy.

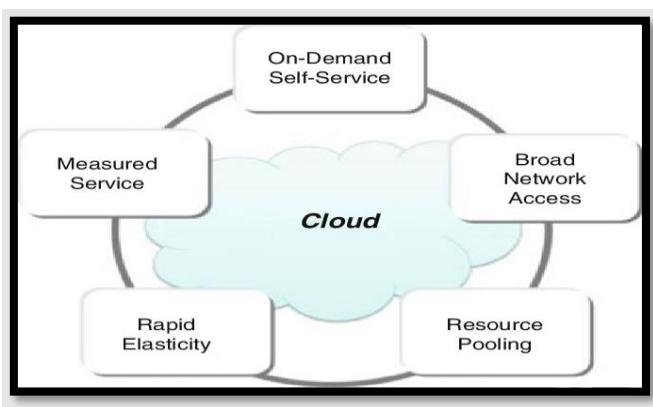


Image: Essential Characteristics of Cloud Computing

Reference: [https://www.researchgate.net/profile/Kevin-Curran/publication/314374762/figure/fig1/AS:596455630860288@1519217512203/T he-five-essential-characteristics-of-the-cloud-computing-model-The-Open-group-2013.png](https://www.researchgate.net/profile/Kevin-Curran/publication/314374762/figure/fig1/AS:596455630860288@1519217512203/T-he-five-essential-characteristics-of-the-cloud-computing-model-The-Open-group-2013.png)

### Resource pooling

- Several users use the same pool of computing infrastructure.
- Important concept for energy saving.
- Not every one of the users gets a dedicated computer.

- But it is possible through cloud providers to gain access to single dedicated machines, sometimes called bare metal or dedicated instance.
- Consequently, this is not in practice an essential characteristic.

#### 7.2.4 Cloud Advantages

- **Scalability** - Get the specific amount of power you need, when you need it, enabling you to increase and decrease levels to suit your businesses demands.
- **Cost savings** - Thanks to the utility pricing model of the cloud, you only pay for what you use. Avoid upfront hardware costs, as well as the costs of maintenance, software upgrades, power, and the manpower to manage it all.
- **Disaster Recovery** - A full back-up solution of not just your data but your entire server operating system and applications.
- **Accessibility** - Host all your data and systems via a secure leased line connection which provides a high-speed private link between you and your provider.
- **Resilience** - Protecting your business against any potential IT failures that could cause down-time or disruption, fully backed-up to provide a complete disaster recovery solution.
- **Business Focus** - When you rely on the cloud, you can apply your time and money towards your business priorities, rather than worrying about your IT infrastructure.

### 7.3 Cloud Deployment & Service Delivery Models

Cloud is the future of computing. It is about outsourcing of IT services and infrastructure to make them accessible remotely via the Internet. Utilizing cloud-computing models boosts not only productivity but also provide a competitive edge to organizations. The growing popularity of cloud computing has given rise to different types of cloud service deployment models and strategies. Therefore, today there exists a variety of enterprise cloud solutions depending on the degree of desired outsourcing needs. It is along with their customization flexibility, control, and data management within the organization. Further, it involves the pooling of specialized human and technical resources to effectively manage existing systems and applications as it helps in meeting the requirements of organizations and users.

#### 7.3.1 Different Types of Cloud Computing Deployment Models

Most cloud hubs have tens of thousands of servers and storage devices to enable fast loading. It is often possible to choose a geographic area to put the data “closer” to users. Thus, deployment models of cloud computing are categorized based on their location. To know which deployment model would best fit the requirements of your organization, let us first learn about the types of cloud deployment models.

- Private Cloud** - It is a cloud-based infrastructure used by stand-alone organizations. It offers greater control over security. The data is backed up by a firewall and internally, and can be hosted internally or externally. Private clouds are perfect for organizations that have high-security requirements, high management demands, and availability requirements.
- Community Cloud** - It is a mutually shared model between organizations that belong to a particular community such as banks, government organizations, or commercial enterprises. Community members generally share similar issues of privacy, performance, and security. This type of deployment model of cloud computing is managed and hosted internally or by a third-party vendor.

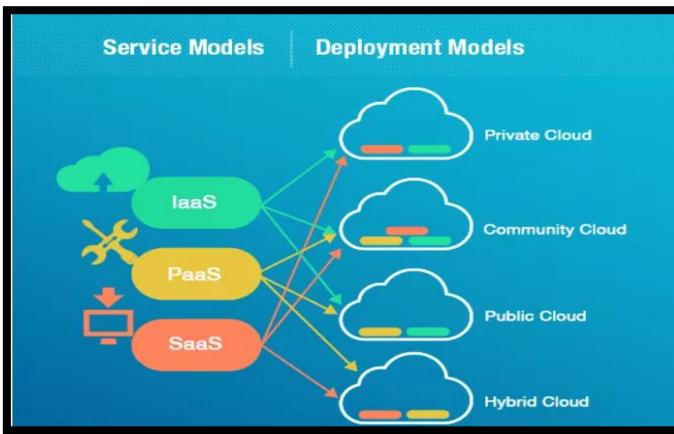


Image: Cloud Service Models

Reference: <https://www.oreilly.com/library/view/the-enterprise-cloud/>

- Public Cloud** - This type of cloud services is provided on a network for public use. Customers have no control over the location of the infrastructure. It is based on a shared cost model for all the users, or in the form of a licensing policy such as pay per user. Public deployment models in the cloud are perfect for organizations with growing and fluctuating demands. It is also popular among businesses of all sizes for their web applications, webmail, and storage of non-sensitive data.
- Hybrid Cloud** - This model incorporates the best of both private and public clouds, but each can remain as separate entities. Further, as part of this deployment of cloud computing model, the internal, or external providers can provide resources. A hybrid cloud is ideal for scalability, flexibility, and security. A perfect example of this scenario would be that of an organization who uses the private cloud to secure their data and interacts with its customers using the public cloud.

### 7.3.2 Cloud Service Delivery Models

There are the following three types of cloud service models -

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS)

### *Infrastructure as a Service (IaaS)*

IaaS is also known as **Hardware as a Service (HaaS)**. It is a computing infrastructure managed over the internet. The main advantage of using IaaS is that it helps users to avoid the cost and complexity of purchasing and managing the physical servers.

### **Characteristics of IaaS**

There are the following characteristics of IaaS -

- Resources are available as a service
- Services are highly scalable
- Dynamic and flexible
- GUI and API-based access
- Automated administrative tasks

**Example:** Microsoft Azure, DigitalOcean, Linode, Amazon Web Services (AWS), Google Compute Engine (GCE), Rackspace, and Cisco Metacloud.

IaaS provider provides the following services –

1. **Compute:** Computing as a Service includes virtual central processing units and virtual main memory for the VMs that is provisioned to the end-users.
2. **Storage:** IaaS provider provides back-end storage for storing files.
3. **Network:** Network as a Service (NaaS) provides networking components such as routers, switches, and bridges for the VMs.
4. **Load balancers:** It provides load balancing capability at the infrastructure layer.

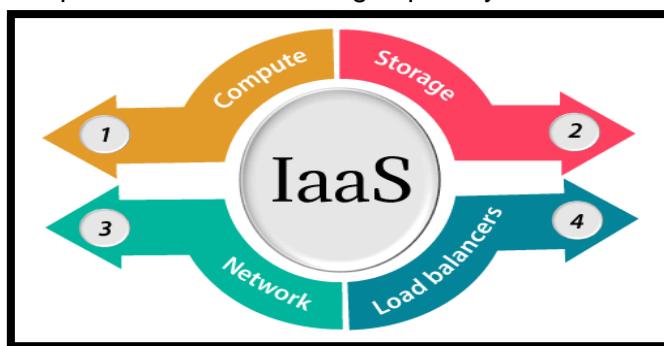


Image: Cloud IaaS model components

Reference: [https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRWUvBkbN6R\\_V7theCeoWuijOajhOEoW61\\_6g&usqp=CAU](https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRWUvBkbN6R_V7theCeoWuijOajhOEoW61_6g&usqp=CAU)

### **Advantages of IaaS cloud computing layer**

There are the following advantages of IaaS computing layer -

1. **Shared infrastructure** - IaaS allows multiple users to share the same physical infrastructure.
2. **Web access to the resources** IaaS allows IT users to access resources over the internet.

- 3. Pay-as-per-use model** - IaaS providers provide services based on the pay-as-per-use basis. The users are required to pay for what they have used.
- 4. Focus on the core business** - IaaS providers focus on the organization's core business rather than on IT infrastructure.

### **Disadvantages of IaaS cloud computing layer**

- 1. Security** - Security is one of the biggest issues in IaaS. Most of the IaaS providers are not able to provide 100% security.
- 2. Maintenance & Upgrade** - Although IaaS service providers maintain the software, but they do not upgrade the software for some organizations.
- 3. Interoperability issues** - It is difficult to migrate VM from one IaaS provider to the other, so the customers might face problem related to vendor lock-in.

### **Some important point about IaaS cloud computing layer.**

IaaS cloud computing platform cannot replace the traditional hosting method, but it provides more than that, and each resource which are used are predictable as per the usage. IaaS cloud computing platform may not eliminate the need for an in-house IT department. It will be needed to monitor or control the IaaS setup.

Breakdowns at the IaaS cloud computing platform vendor's can bring your business to the halt stage. Assess the IaaS cloud computing platform vendor's stability and finances. Make sure that SLAs (i.e., Service Level Agreement) provide backups for data, hardware, network, and application failures. Image portability and third-party support is a plus point. The IaaS cloud computing platform vendor can get access to your sensitive data. So, engage with credible companies or organizations. Study their security policies and precautions.

#### *Platform as a Service (PaaS)*

PaaS cloud computing platform is created for the programmer to develop, test, run, and manage the applications.

### **Characteristics of PaaS**

There are the following characteristics of PaaS -

- Accessible to various users via the same development application.
- Integrates with web services and databases.
- Builds on virtualization technology, so resources can easily be scaled up or down as per the organization's need.
- Supports multiple languages and frameworks.
- Provides an ability to "Auto-scale".

**Example:** AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos, Magento Commerce Cloud, and OpenShift.

## **Platform as a Service | PaaS**

Platform as a Service (PaaS) provides a runtime environment. It allows programmers to easily create, test, run, and deploy web applications. You can purchase these applications from a cloud service provider on a pay-as-per use basis and access them using the Internet connection. In PaaS, back-end scalability is managed by the cloud service provider, so end-users do not need to worry about managing the infrastructure.

PaaS includes infrastructure (servers, storage, and networking) and platform (middleware, development tools, database management systems, business intelligence, and more) to support the web application life cycle.

**Example:** Azure App Service, Google App Engine, Force.com, Joyent.

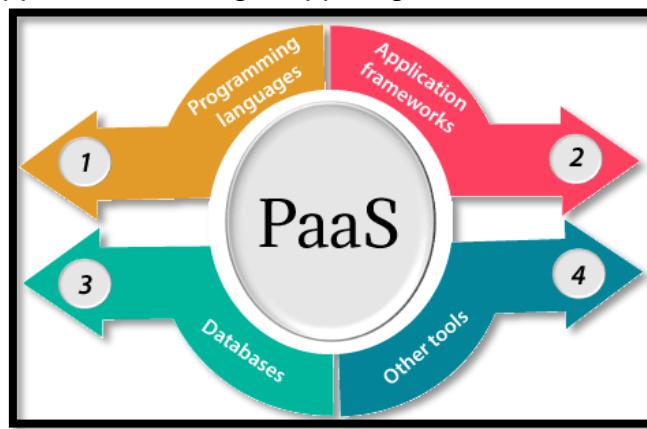


Image: Cloud PaaS Components  
Reference: <https://encrypted-tbn0.gstatic.com/images>

### **Programming languages**

PaaS providers provide various programming languages for the developers to develop the applications. Some popular programming languages provided by PaaS providers are Java, PHP, Ruby, Perl, and Go.

### **Application frameworks**

PaaS providers provide application frameworks to easily understand the application development. Some popular application frameworks provided by PaaS providers are Node.js, Drupal, Joomla, WordPress, Spring, Play, Rack, and Zend.

### **Databases**

PaaS providers provide various databases such as ClearDB, PostgreSQL, MongoDB, and Redis to communicate with the applications.

## Other tools

PaaS providers provide various other tools that are required to develop, test, and deploy the applications.

## Advantages of PaaS

There are the following advantages of PaaS -

1. **Simplified Development** - PaaS allows developers to focus on development and innovation without worrying about infrastructure management.
2. **Lower risk** - No need for up-front investment in hardware and software. Developers only need a PC and an internet connection to start building applications.
3. **Prebuilt business functionality**- Some PaaS vendors also provide already defined business functionality so that users can avoid building everything from scratch and hence can directly start the projects only.
4. **Instant community**- PaaS vendors frequently provide online communities where the developer can get the ideas to share experiences and seek advice from others.

## Disadvantages of PaaS cloud computing layer

1. **Vendor lock-in**- One has to write the applications according to the platform provided by the PaaS vendor, so the migration of an application to another PaaS vendor would be a problem.
2. **Data Privacy**- Corporate data, whether it can be critical or not, will be private, so if it is not located within the walls of the company, there can be a risk in terms of privacy of data.
3. **Integration with the rest of the systems applications**- It may happen that some applications are local, and some are in the cloud. So, there will be chances of increased complexity when we want to use data which in the cloud with the local data.

### *Software as a Service (SaaS)*

SaaS is also known as "**on-demand software**". It is a software in which the applications are hosted by a cloud service provider. Users can access these applications with the help of internet connection and web browser.

## Characteristics of SaaS

There are the following characteristics of SaaS -

- Managed from a central location
- Hosted on a remote server
- Accessible over the internet

- Users are not responsible for hardware and software updates. Updates are applied automatically.

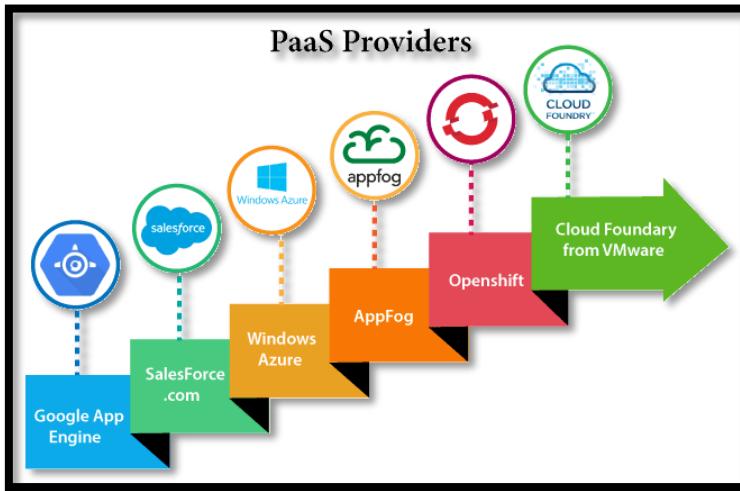


Image: PaaS Providers

## Software as a Service | SaaS

SaaS is also known as "**On-Demand Software**". It is a software distribution model in which services are hosted by a cloud service provider. These services are available to end-users over the internet so, the end-users do not need to install any software on their devices to access these services.

There are the following services provided by SaaS providers -

- 1. Business Services** - SaaS Provider provides various business services to start-up the business. The SaaS business services include **ERP** (Enterprise Resource Planning), **CRM** (Customer Relationship Management), **billing**, and **sales**.
- 2. Document Management** - SaaS document management is a software application offered by a third party (SaaS providers) to create, manage, and track electronic documents. **Example:** Slack, Samepage, Box, and Zoho Forms.
- 3. Social Networks** - As we all know, social networking sites are used by the general public, so social networking service providers use SaaS for their convenience and handle the general public's information.
- 4. Mail Services** - To handle the unpredictable number of users and load on e-mail services, many e-mail providers offering their services using SaaS.



Image: Cloud SaaS Model  
 Reference: <https://encrypted-tbn0.gstatic.com/images>

## Advantages of SaaS cloud computing layer

- 1. SaaS is easy to buy** - SaaS pricing is based on a monthly fee or annual fee subscription, so it allows organizations to access business functionality at a low cost, which is less than licensed applications. Unlike traditional software, which is sold as a licensed based with an up-front cost (and often an optional ongoing support fee), SaaS providers are generally pricing the applications using a subscription fee, most commonly a monthly or annually fee.
- 2. One to Many** - SaaS services are offered as a one-to-many model means a single instance of the application is shared by multiple users.
- 3. Less hardware required for SaaS** - The software is hosted remotely, so organizations do not need to invest in additional hardware.
- 4. Low maintenance required for SaaS** - Software as a service removes the need for installation, set-up, and daily maintenance for the organizations. The initial set-up cost for SaaS is typically less than the enterprise software. SaaS vendors are pricing their applications based on some usage parameters, such as a number of users using the application. So, SaaS does easy to monitor and automatic updates.
- 5. No special software or hardware versions required** - All users will have the same version of the software and typically access it through the web browser. SaaS reduces IT support costs by outsourcing hardware and software maintenance and support to the IaaS provider.
- 6. Multidevice support** - SaaS services can be accessed from any device such as desktops, laptops, tablets, phones, and thin clients.
- 7. No client-side installation** - SaaS services are accessed directly from the service provider using the internet connection, so do not need to require any software installation.

## Disadvantages of SaaS cloud computing layer

- 1. Security** - Actually, data is stored in the cloud, so security may be an issue for some users. However, cloud computing is not more secure than in-house deployment.
- 2. Latency issue** - Since data and applications are stored in the cloud at a variable distance from the end-user, there is a possibility that there may be greater latency when interacting with the application compared to local deployment. Therefore, the SaaS model is not suitable for applications whose demand response time is in milliseconds.
- 3. Total Dependency on Internet** - Without an internet connection, most SaaS applications are not usable.
- 4. Switching between SaaS vendors is difficult** - Switching SaaS vendors involve the difficult and slow task of transferring the very large data files over the internet and then converting and importing them into another SaaS also.

### 7.3.3 Difference between IaaS, PaaS, and SaaS

The below table 1 shows the difference between IaaS, PaaS, and SaaS  
Table 1

IaaS	PaaS	SaaS
It provides a virtual data center to store information and create platforms for app development, testing, and deployment.	It provides virtual platforms and tools to create, test, and deploy apps.	It provides web software and apps to complete business tasks.
It provides access to resources such as virtual machines, virtual storage, etc.	It provides runtime environments and deployment tools for applications.	It provides software as a service to the end-users.
It is used by network architects.	It is used by developers.	It is used by end users.
IaaS provides only Infrastructure.	PaaS provides Infrastructure+Platform.	SaaS provides Infrastructure+Platform +Software.