

TOP 100



Angular INTERVIEW QUESTIONS

HAPPY RAWAT

PREFACE

ABOUT THE BOOK

This book contains 100 very important Angular interview questions.



ABOUT THE AUTHOR

Happy Rawat has around 15 years of experience in software development. He helps candidates in clearing technical interview in tech companies.



Chapters

Basics

1. Angular Framework

2. Components & Modules

3. Data Binding

4. Directives

5. Decorator & Pipes

6. Services & DI

7. Component Lifecycle-Hooks

Advanced

8. Routing

9. Observable\ HttpClient\ RxJS

10. Typescript-Basics

11. Typescript - OOPS

12. Angular Forms

13. Authentication

14. Components Communication

Table of Contents

Chapter 1: Angular Framework

[Q1. What is Angular?](#)

[Q2. What are Angular advantages?](#)

[Q3. What is the difference between AngularJS and Angular?](#)

[Q4. What is NPM?](#)

[Q5. What is CLI tool?](#)

[Q6. What is Typescript? What are the advantages of Typescript over Javascript?](#)

[Q7. Where to store static files in Angular project?](#)

[Q8. What is the role of Angular.json file in Angular?](#)

[Q9. What is the difference between JIT and AOT in Angular?](#)

Chapter 2: Components & Modules

[Q10. What are Components in Angular?](#)

[Q11. What is a Selector and Template?](#)

[Q12. What is Module in Angular? What is app.module.ts file?](#)

[Q13. How an Angular App gets Loaded and Started? What are index.html, app-root, selector and main.ts?](#)

[Q14. What is a Bootstrapped Module & Bootstrapped Component?](#)

Chapter 3: Data Binding

[Q15. What is Data Binding in Angular?](#)

[Q16. What is String Interpolation in Angular?](#)

[Q17. What is Property Binding in Angular?](#)

[Q18. What is Event Binding in Angular?](#)

[Q19. What is Two way Binding in Angular?](#)

Chapter 4: Directives

[Q20. What are Directives? What are the type of directives?](#)

[Q21. What is *ngIf Structural directive?](#)

[Q22. What is *ngFor Structural directive?](#)

[Q23. What is *ngSwitch Structural directive?](#)

[Q24. What is \[ngStyle\] Attribute directive?](#)

[Q25. What is \[ngClass\] Attribute directive?](#)

[Q26. What is the difference between Component, Attribute and Structural Directives?](#)

Chapter 5: Decorator & Pipes

[Q27. What is Decorator?](#)

[Q28. What are the types of Decorator?](#)

[Q29. What are Pipes? What are the types of Pipes & Parameterized Pipes?](#)

[Q30. What is Chaining Pipes?](#)

Chapter 6: Services & Dependency Injection

[Q31. Explain Services with Example?](#)

[Q32. How to create Service in Angular?](#)

[Q33. How to use Dependency Injector with Services in Angular?](#)

[Q34. What is Hierarchical Dependency Injection?](#)

[Q35. What is Provider in Angular?](#)

[Q36. What is the role of @Injectable Decorator in a Service?](#)

Chapter 7: Decorators & Lifecycle-Hooks

[Q37. What are Parent-Child Components?](#)

[Q38. What is @Input Decorator? How to transfer data from Parent component to Child component?](#)

[Q39. What is @Output Decorator and Event Emitter?](#)

[Q40. What are Lifecycle Hooks in Angular?](#)

[Q41. What is a Constructor in Angular?](#)

[Q42. What is ngOnChanges life cycle hook in Angular?](#)

[Q43. What is ngOnInit life cycle hook in Angular?](#)

[Q44. What is the difference between constructor and ngOnInit?](#)

Chapter 8: Routing

[Q45. What is Routing? How to setup Routing?](#)

[Q46. What is router outlet?](#)

[Q47. What are router links?](#)

Chapter 9: Observable\ HttpClient\ RxJS

[Q48. What are Asynchronous operations?](#)

[Q49. What is the difference between Promise and Observable?](#)

[Q50. What is RxJS?](#)

[Q51. What is Observable? How to implement Observable ?](#)

[Q52. What is the role of HttpClient in Angular?](#)

[Q53. What are the steps for fetching the data with HttpClient & Observable?](#)

[Q54. How to do HTTP Error Handling in Angular?](#)

Chapter 10: Typescript-Basics

[Q55. What is Typescript? Or What is the difference between Typescript and Javascript?](#)

[Q56. How to install Typescript and check version?](#)

[Q57. What is the difference between let and var keyword?](#)

[Q58. What is Type annotation?](#)

[Q59. What are Built in/ Primitive and User-Defined/ Non-primitive Types in Typescript?](#)

[Q60. What is “any” type in Typescript?](#)

[Q61. What is Enum type in Typescript?](#)

[Q62. What is the difference between void and never types in Typescript?](#)

[Q63. What is Type Assertion in Typescript?](#)

[Q64. What are Arrow Functions in Typescript?](#)

Chapter 11: Typescript - OOPS

[Q65. What is Object Oriented Programming in Typescript?](#)

[Q66. What are Classes and Objects in Typescript?](#)

[Q67. What is Constructor?](#)

[Q68. What are Access Modifiers in Typescript?](#)

[Q69. What is Encapsulation in Typescript?](#)

[Q70. What is Inheritance in Typescript?](#)

[Q71. What is Polymorphism in Typescript?](#)

[Q72. What is Interface in Typescript?](#)

[Q73. What's the difference between extends and implements in TypeScript ?](#)

[Q74. Is Multiple Inheritance possible in Typescript?](#)

Chapter 12: Angular Forms

[Q75. What are Angular Forms? What are the type of Angular Forms?](#)

[Q76. What is the difference between Template Driven Forms & Reactive Forms?](#)

[Q77. How to setup Template Driven Forms?](#)

[Q78. How to apply Required field validation in template driven forms?](#)

[Q79. What is Form Group and Form Control in Angular?](#)

[Q80. How to setup Reactive Forms?](#)

[Q81. How to do validations in reactive forms?](#)

Chapter 13: Authentication/ JWT/ Auth Gurad/ HTTP Interceptor

[Q82. What is Authentication & Authorization in Angular?](#)

[Q83. What is JWT Token Authentication in Angular?](#)

[Q84. How to Mock or Fake an API for JWT Authentication?](#)

[Q85. How to implement the Authentication with JWT in Angular?](#)

[Q86. What is Auth Guard?](#)

[Q87. What is HTTP Interceptor?](#)

[Q88. How to Retry automatically if there is an error response from API?](#)

[Q89. What are the parts of JWT Token?](#)

[Q90. What is Postman?](#)

[Q91. Which part of the request has the token stored when sending to API?](#)

Chapter 14: Components Communication

[Q92. What are the various ways to communicate between the components?](#)

[Q93. What is ContentProjection? What is <ng-content>?](#)

[Q94. What is Template Reference Variable in Angular?](#)

[Q95. What is the role of ViewChild in Angular?](#)

[Q96. How to access the child component from parent component with ViewChild?](#)

[Q97. What is the difference between ViewChild and ViewChildren? What is QueryList?](#)

[Q98. What is ContentChild?](#)

[Q99. What is the difference between ContentChild & ContentChildren?](#)

[Q100. Compare ng-Content, ViewChild, ViewChildren, ContentChild & ContentChildren?](#)



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 1: Angular Framework

Q1. What is **Angular**?

Q2. What are Angular **advantages**?

Q3. What is the difference between **AngularJS** and **Angular**?

Q4. What is **NPM**?

Q5. What is **CLI tool**?

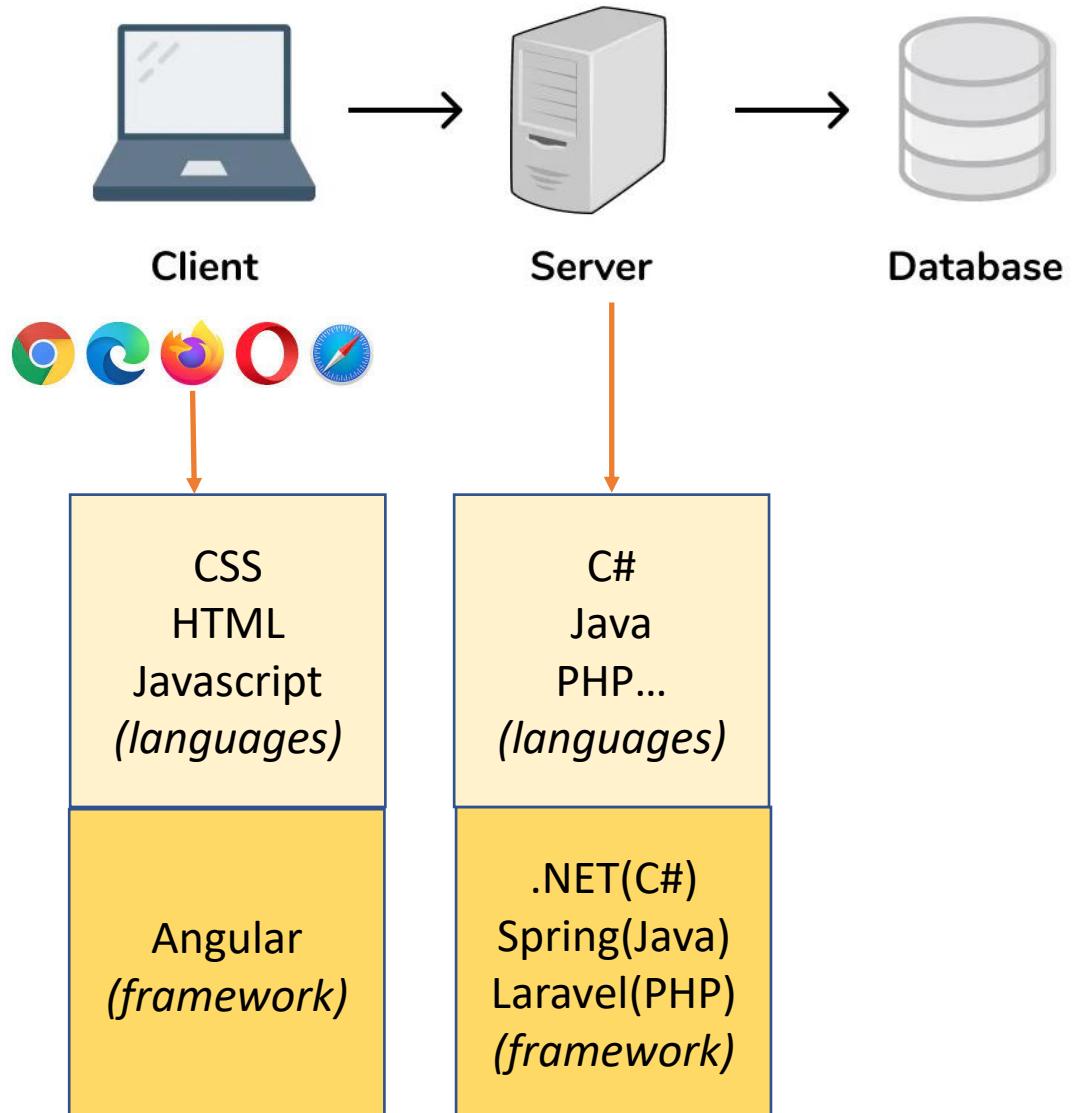
Q6. What is **Typescript**? What are the advantages of Typescript over Javascript?

Q7. Where to store **static files** in Angular project?

Q8. What is the role of **Angular.json** file in Angular?

Q9. What is the difference between **JIT** and **AOT** in Angular?

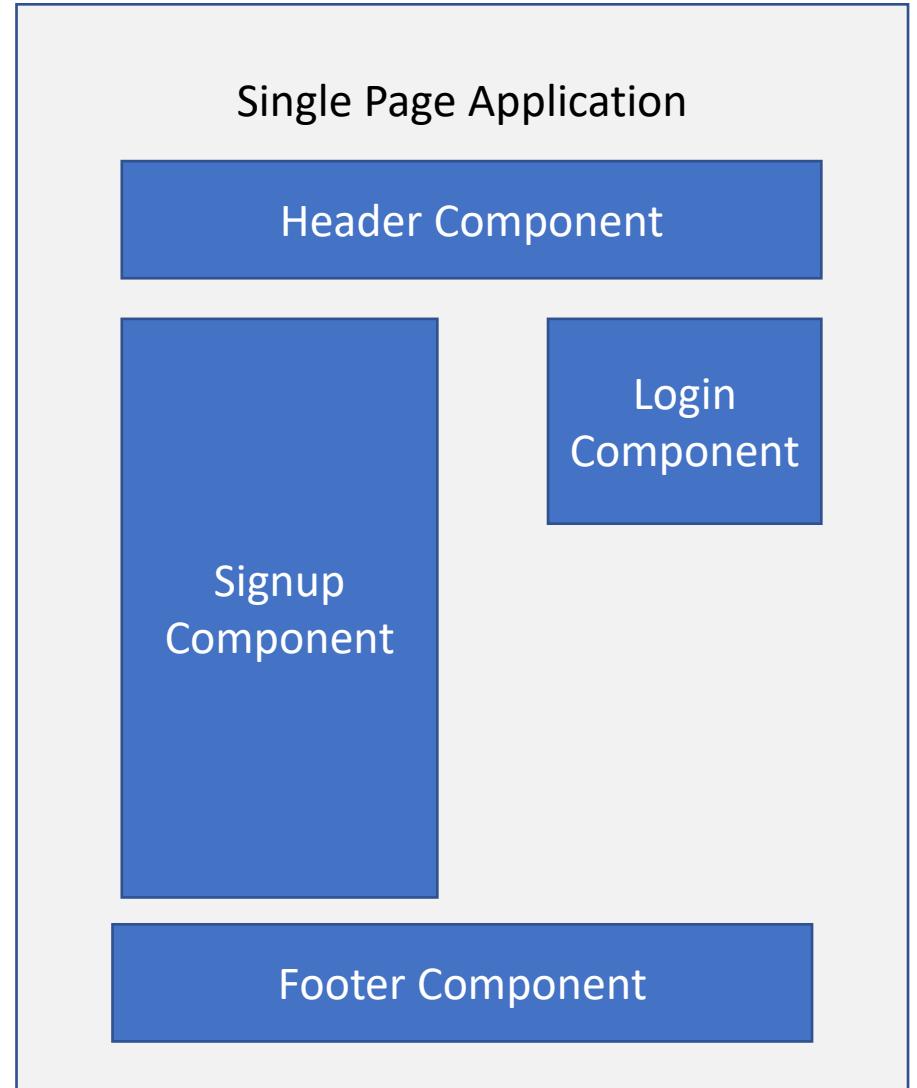
- ❖ Angular is a component-based framework for building structured, scalable and **single page web applications** for client side.
- ❖ Like we have .NET, Spring, Laravel framework for Server side programming. Similarly we have Angular framework for Client side programming.



❖ What is single page application?

In Single Page Application, we only have one page and the body content of that page will be updated as per the request.

For example, in the image we have single page with multiple components. These components will be replaced by the new components to display the data but the page will remain same.





1. It is Relatively simple to build Single Page Applications (by using Components)



2. Flexible and structured client applications (by using OOPS concepts)



3. Angular is cross platform and open source(Free to use)

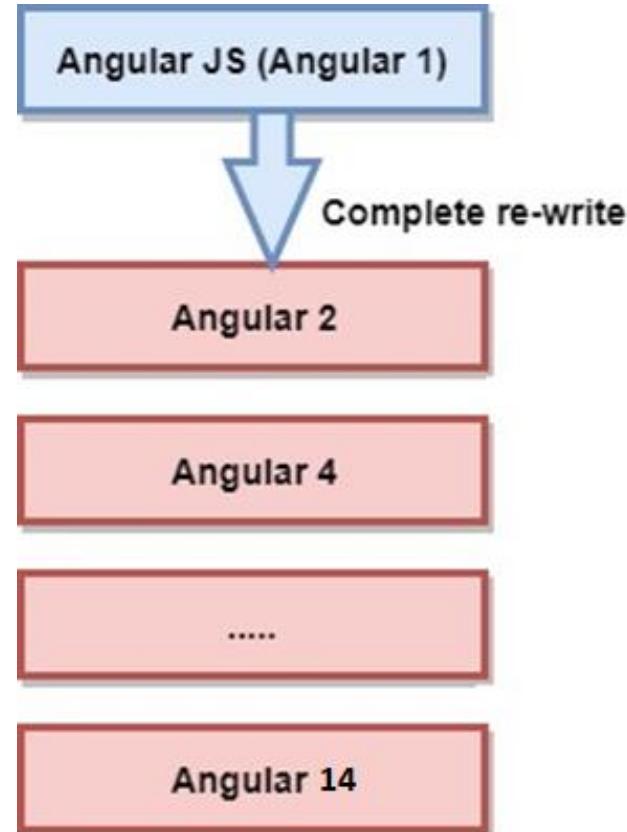


4. Writing Reusable Code is easy (by using Services)



5. Testing is easy (by using spec.ts)

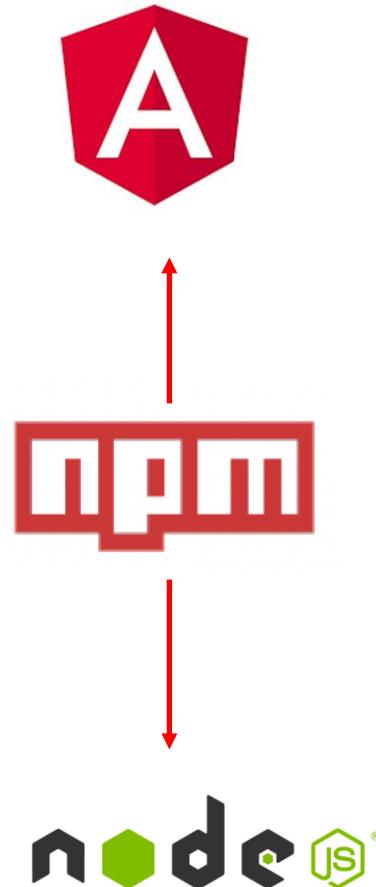
- ❖ In the image, you can see Angular History from Angular JS to Angular 14 and more....

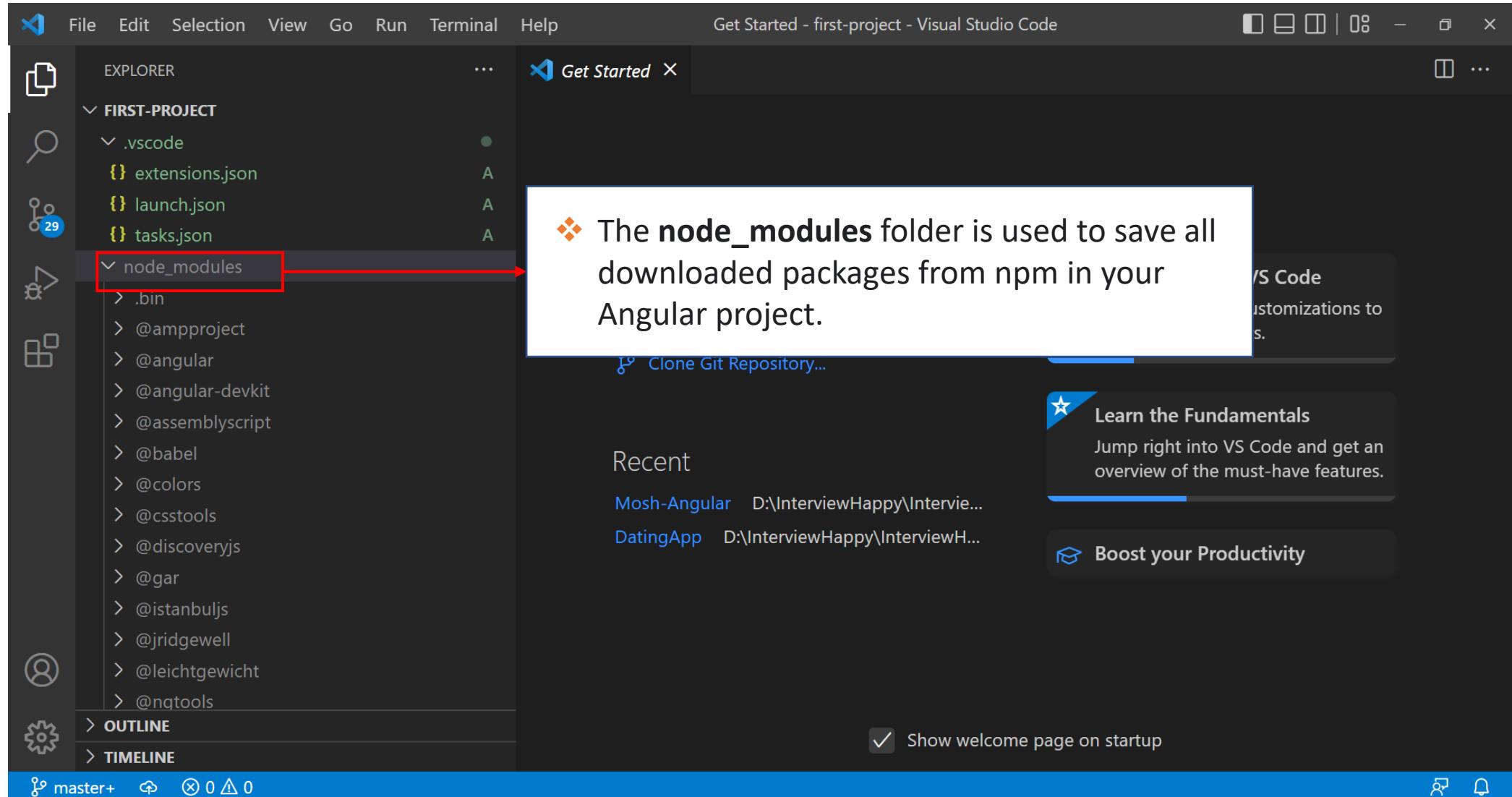




Angular JS	Angular
1. It only supports JavaScript.	It support both JavaScript and TypeScript.
2. This framework has a model-view-controller (MVC) architecture.	This framework has a component based architecture.
3. It does not have CLI tool.	It has CLI tool.
4. It does not use Dependency Injection.	It uses Dependency Injection.
5. It does not support mobile browsers.	It also support mobile browsers.
6. It is not so fast	It is very fast

- ❖ NPM(Node package manager) is an **online repository**, from where you can get thousands of **free libraries** which can be used in your **angular** or node js projects.

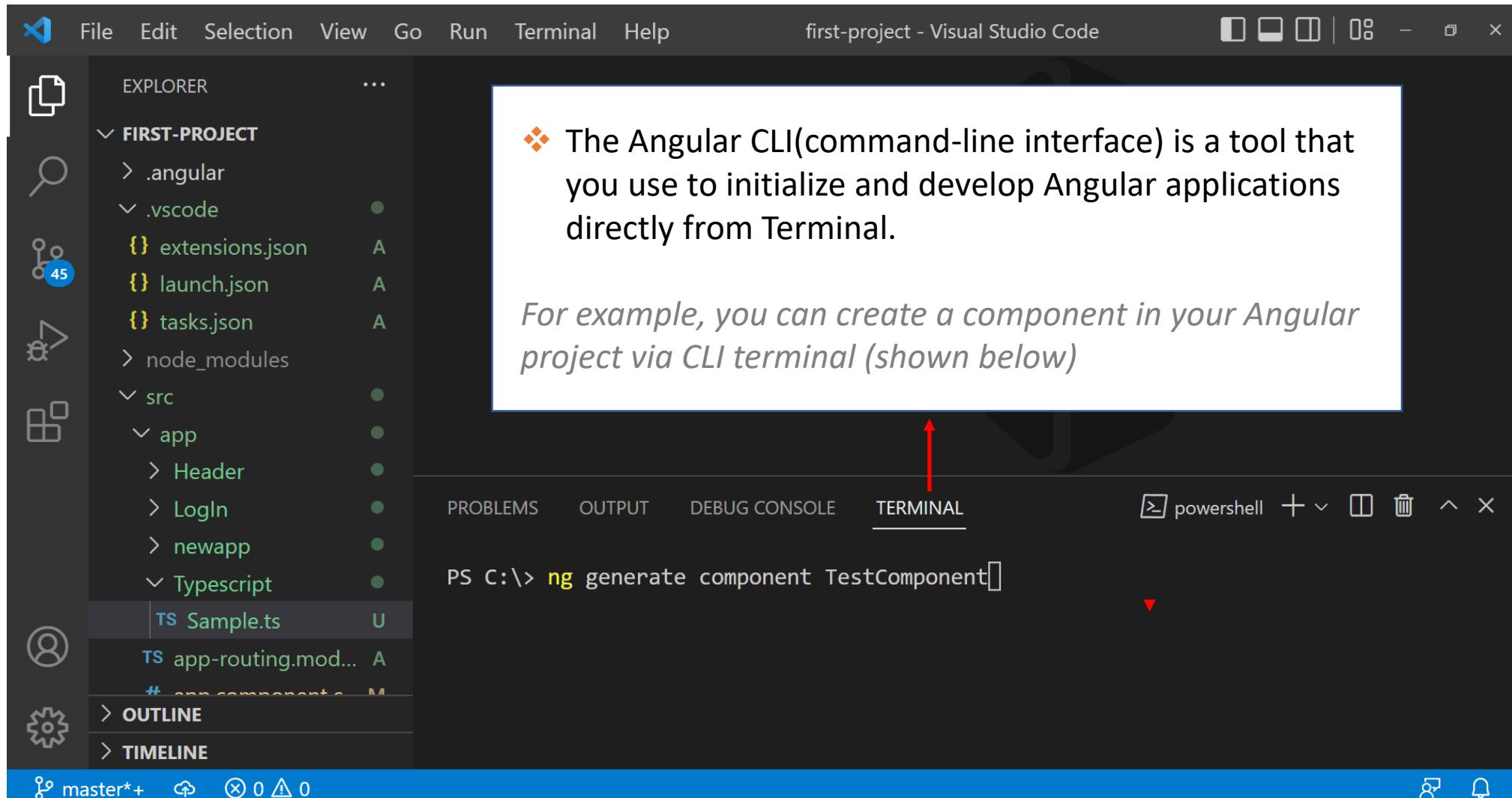




The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left displays a project structure for 'FIRST-PROJECT'. A red arrow points from the text in the callout to the 'node_modules' folder in the Explorer. The main 'Get Started' screen in the center contains a callout with the following text:

❖ The **node_modules** folder is used to save all downloaded packages from npm in your Angular project.

The 'node_modules' folder is highlighted with a red box in the Explorer sidebar. The 'Get Started' screen also features sections for 'Recent' projects and 'Learn the Fundamentals' and 'Boost your Productivity' guides.



The screenshot shows the Visual Studio Code interface with the title "first-project - Visual Studio Code". The Explorer sidebar on the left displays the project structure of "FIRST-PROJECT" with files like ".angular", ".vscode", "extensions.json", "launch.json", "tasks.json", "node_modules", "src", "app", "Header", "LogIn", "newapp", "TypeScript", "Sample.ts", "app-routing.mod...", and "app-components...". The Terminal tab is selected at the bottom, showing the command "PS C:\> ng generate component TestComponent". A callout box with a red arrow points from the text "you use to initialize and develop Angular applications directly from Terminal." to the "TERMINAL" tab. The callout box contains the following text:

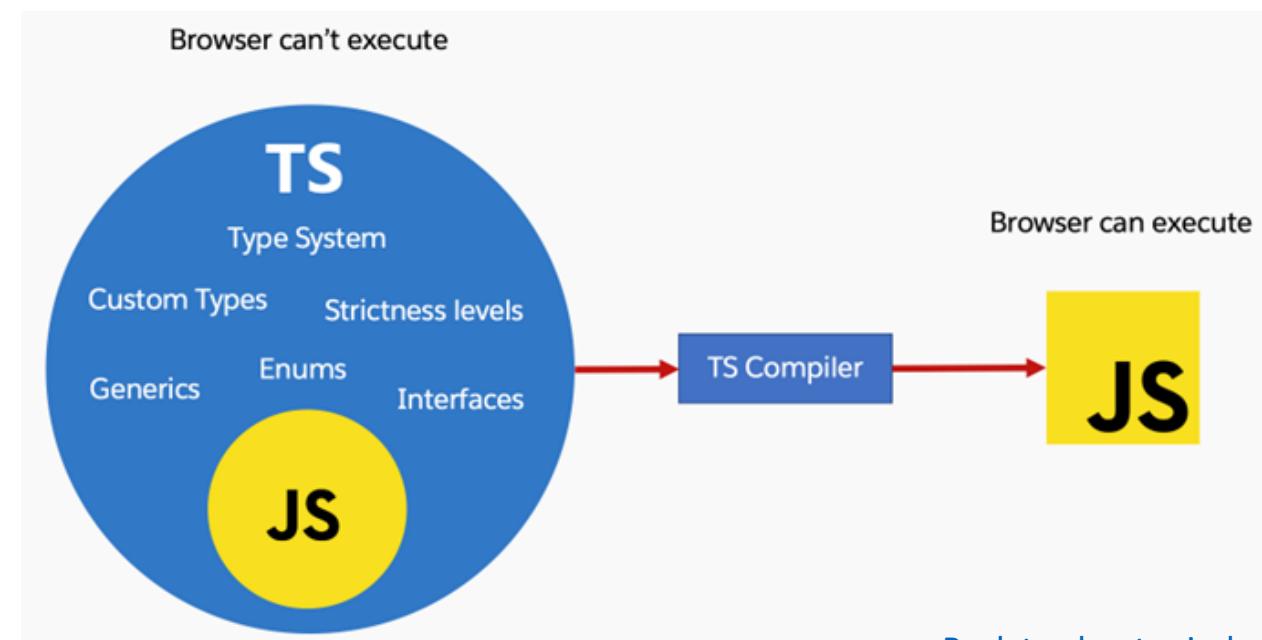
- ❖ The Angular CLI(command-line interface) is a tool that you use to initialize and develop Angular applications directly from Terminal.

For example, you can create a component in your Angular project via CLI terminal (shown below)

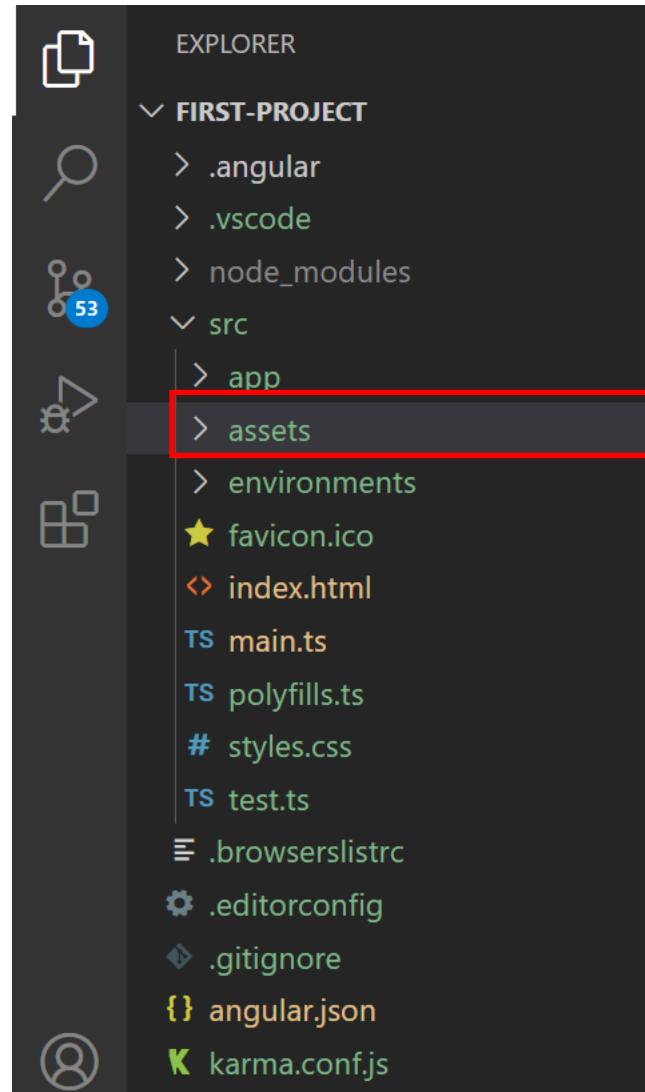
- ❖ Typescript is an open-source programming language.
It's advantage are:

1. Typescript is a strongly typed language.
2. Typescript is a superset of JavaScript.
3. It has Object oriented features.
4. Detect error at compile time.

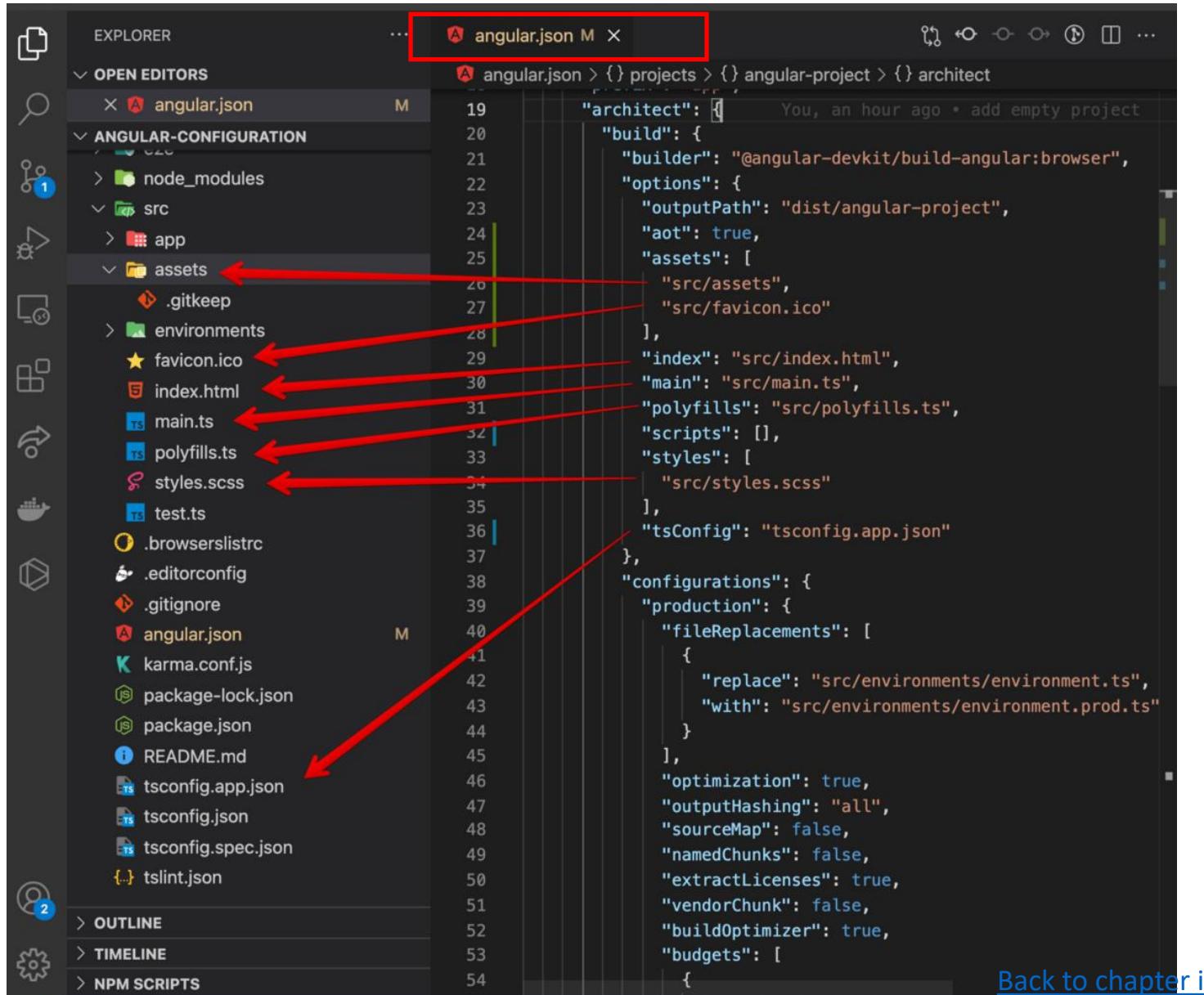
- ❖ In image, you can see TS(Typescript) contains JS(Javascript) and other additional features(type system, custom types, generics, enums, interfaces).
- ❖ Browser can't execute typescript, so finally TS Compiler will convert the TS to JS only, which browser can understand.



- ❖ Store static files in **assets folder** of the Angular project.



- ❖ The angular.json file is the **primary configuration** file for an Angular project.
- ❖ *Configuration file means, in image you see the angular.json file contains all the links of all the files in Angular project.*



```
EXPLORER
  OPEN EDITORS
    angular.json M
  ANGULAR-CONFIGURATION
    node_modules
    src
      app
        assets
          .gitkeep
          environments
            favicon.ico
            index.html
            main.ts
            polyfills.ts
            styles.scss
            test.ts
          .browserslistrc
          .editorconfig
          .gitignore
          angular.json
          karma.conf.js
          package-lock.json
          package.json
          README.md
          tsconfig.app.json
          tsconfig.json
          tsconfig.spec.json
          tslint.json

  angular.json M X
  angular.json > {} projects > {} angular-project > {} architect
  19   "architect": {}
  20     "build": {}
  21       "builder": "@angular-devkit/build-angular:browser",
  22       "options": {}
  23         "outputPath": "dist/angular-project",
  24         "aot": true,
  25         "assets": [
  26           "src/assets",
  27           "src/favicon.ico"
  28         ],
  29         "index": "src/index.html",
  30         "main": "src/main.ts",
  31         "polyfills": "src/polyfills.ts",
  32         "scripts": [],
  33         "styles": [
  34           "src/styles.scss"
  35         ],
  36         "tsConfig": "tsconfig.app.json"
  37     },
  38     "configurations": {
  39       "production": {
  40         "fileReplacements": [
  41           {
  42             "replace": "src/environments/environment.ts",
  43             "with": "src/environments/environment.prod.ts"
  44           }
  45         ],
  46         "optimization": true,
  47         "outputHashing": "all",
  48         "sourceMap": false,
  49         "namedChunks": false,
  50         "extractLicenses": true,
  51         "vendorChunk": false,
  52         "buildOptimizer": true,
  53         "budgets": [
  54           {
  55             "label": "main",
  56             "budget": "512kb"
  57           }
  58         ]
  59       }
  60     }
  61   }
  62 }
```

Back to chapter index

- ❖ Both JIT and AOT are used to compile Angular Typescript components to Javascript, because browser understands Javascript not Typescript.

ANGULAR COMPILE	DETAILS
Just-in-Time (JIT)	Compiles your application in the browser at runtime . This was the default until Angular 8.
Ahead-of-Time (AOT)	Compiles your application and libraries at build time . This is the default starting in Angular 9.

- ❖ All the latest Angular versions use AOT to compile Typescript to Javascript.
- ❖ **Advantage of AOT** - Loading in AOT is much quicker than the JIT, because it already has compiled your code at build time.



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 2: Components & Modules

Q10. What are **Components** in Angular?

Q11. What is a **Selector** and **Template**?

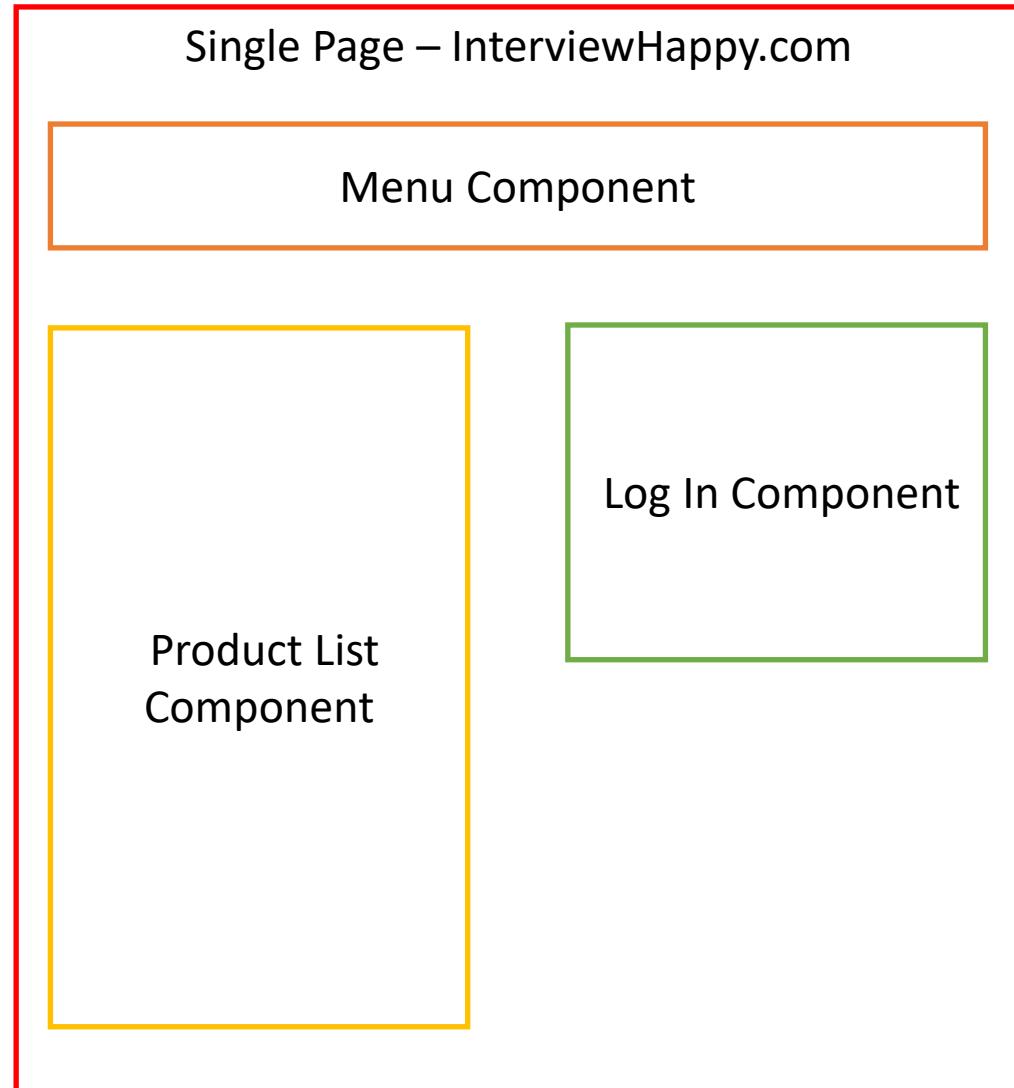
Q12. What is **Module** in Angular? What is **app.module.ts** file?

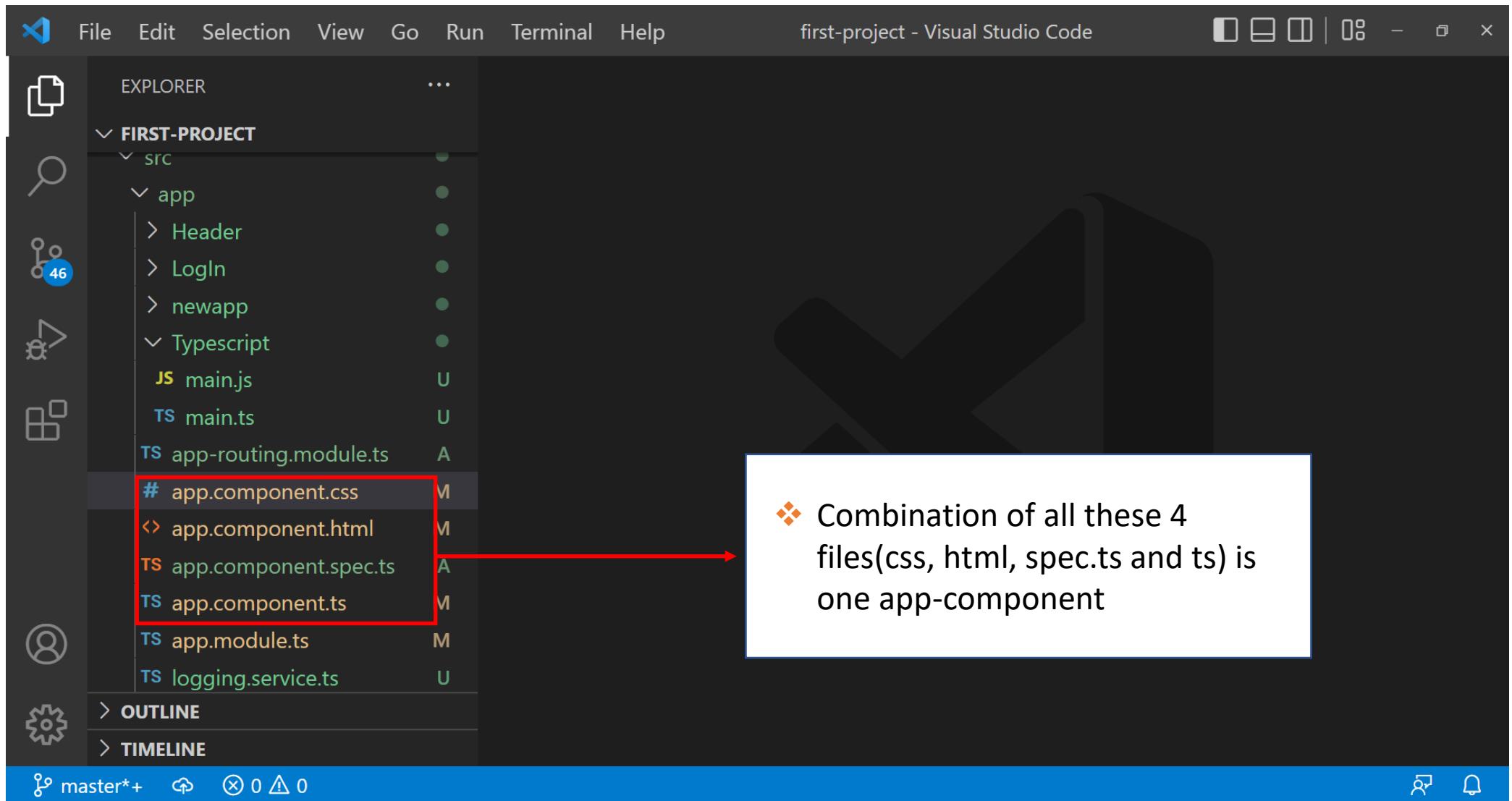
Q13. How an Angular App gets Loaded and Started?
What are **index.html**, **app-root**, **selector** and **main.ts**?

Q14. What is a **Bootstrapped Module** & **Bootstrapped Component**?

- ❖ Components are the most basic UI building block of an Angular app.
- ❖ Remember Angular is a Single Page Application.

For example, in the image we have single page with multiple components. These components will be replaced by the new components to display the data but the page will remain same. So, all these small-small components are like the building block only.





File Edit Selection View Go Run Terminal Help first-project - Visual Studio Code

EXPLORER ...

FIRST-PROJECT

src

app

Header

LogIn

newapp

Typescript

main.js

main.ts

app-routing.module.ts

app.component.css

<> app.component.html

TS app.component.spec.ts

TS app.component.ts

TS app.module.ts

TS logging.service.ts

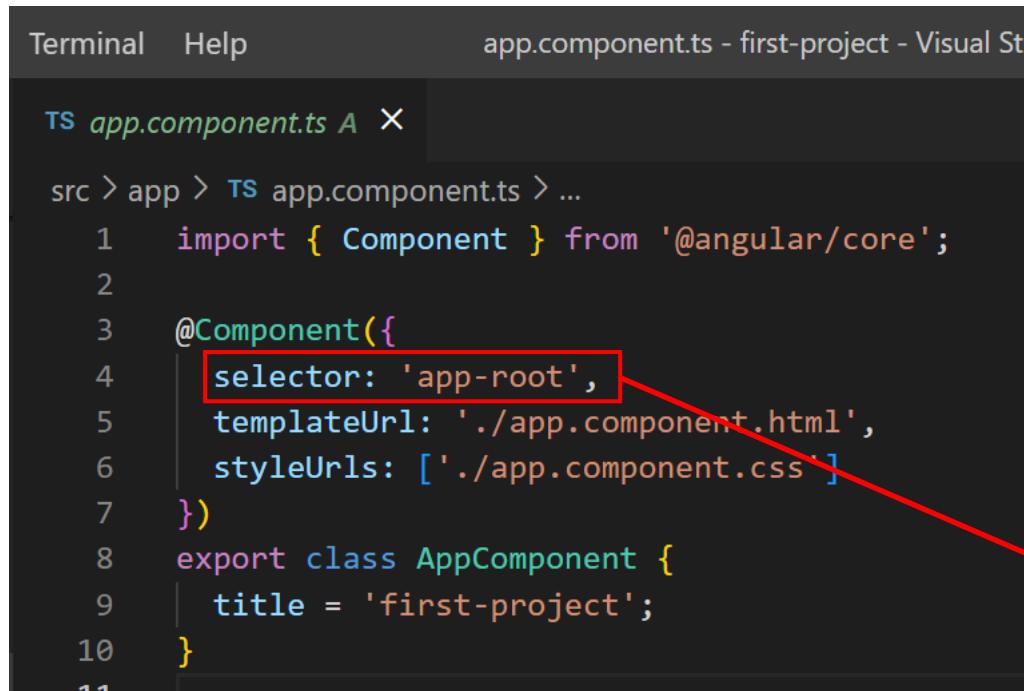
OUTLINE

TIMELINE

master*+ 0 0 △ 0

Combination of all these 4 files(css, html, spec.ts and ts) is one app-component

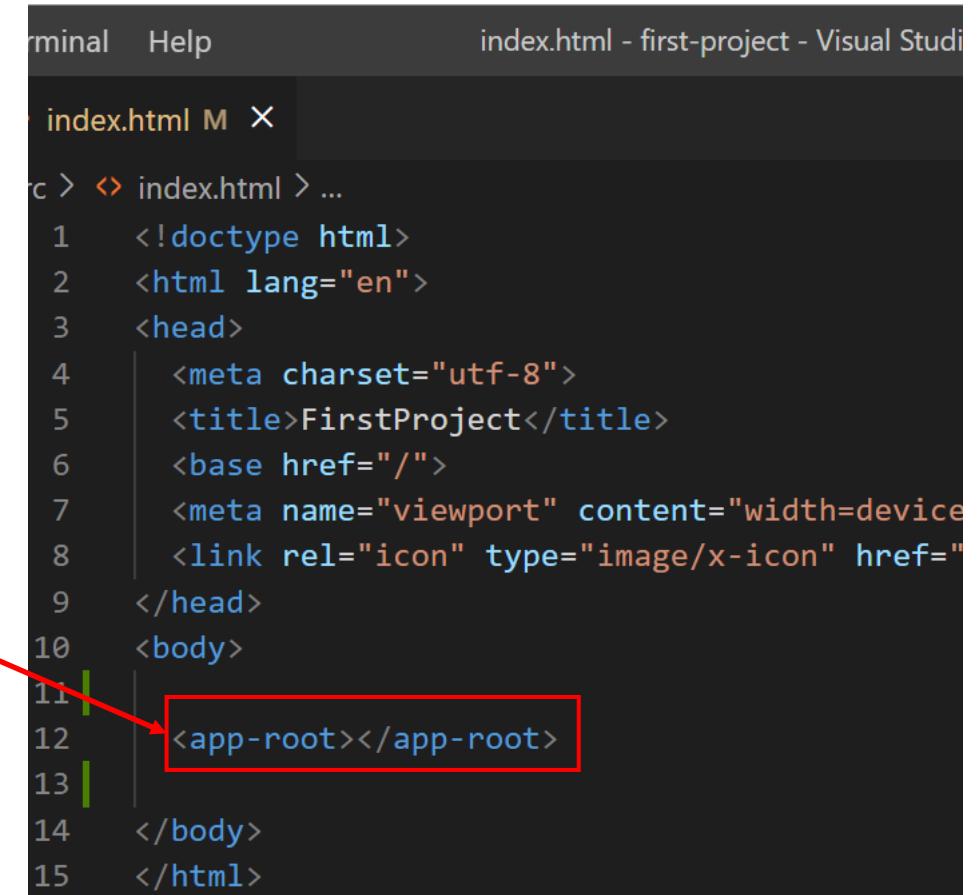
- ❖ A selector is used to **identify each component uniquely** into the component tree.



```
Terminal  Help      app.component.ts - first-project - Visual Studio Code

TS app.component.ts A X

src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root', app-root
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'first-project';
10 }
11
```



```
Terminal  Help      index.html - first-project - Visual Studio Code

index.html M X

src > < app > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>FirstProject</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1.0">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11 <app-root></app-root> app-root
12 </body>
13 </html>
14
15
```

- ❖ In first image, the app-component selector name is “app-root”. And in second picture we are using this selector to position the component in index.html.

- ❖ A Template is a **HTML view** of an Angular component.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<h3>Welcome to Angular Tutorials</h3>`
})
export class AppComponent {
  title = 'MyAngularApp';
}
```

- ❖ And templateUrl is the link of the HTML template file.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-project';
}
```

What is Module in Angular? What is app.module.ts file?

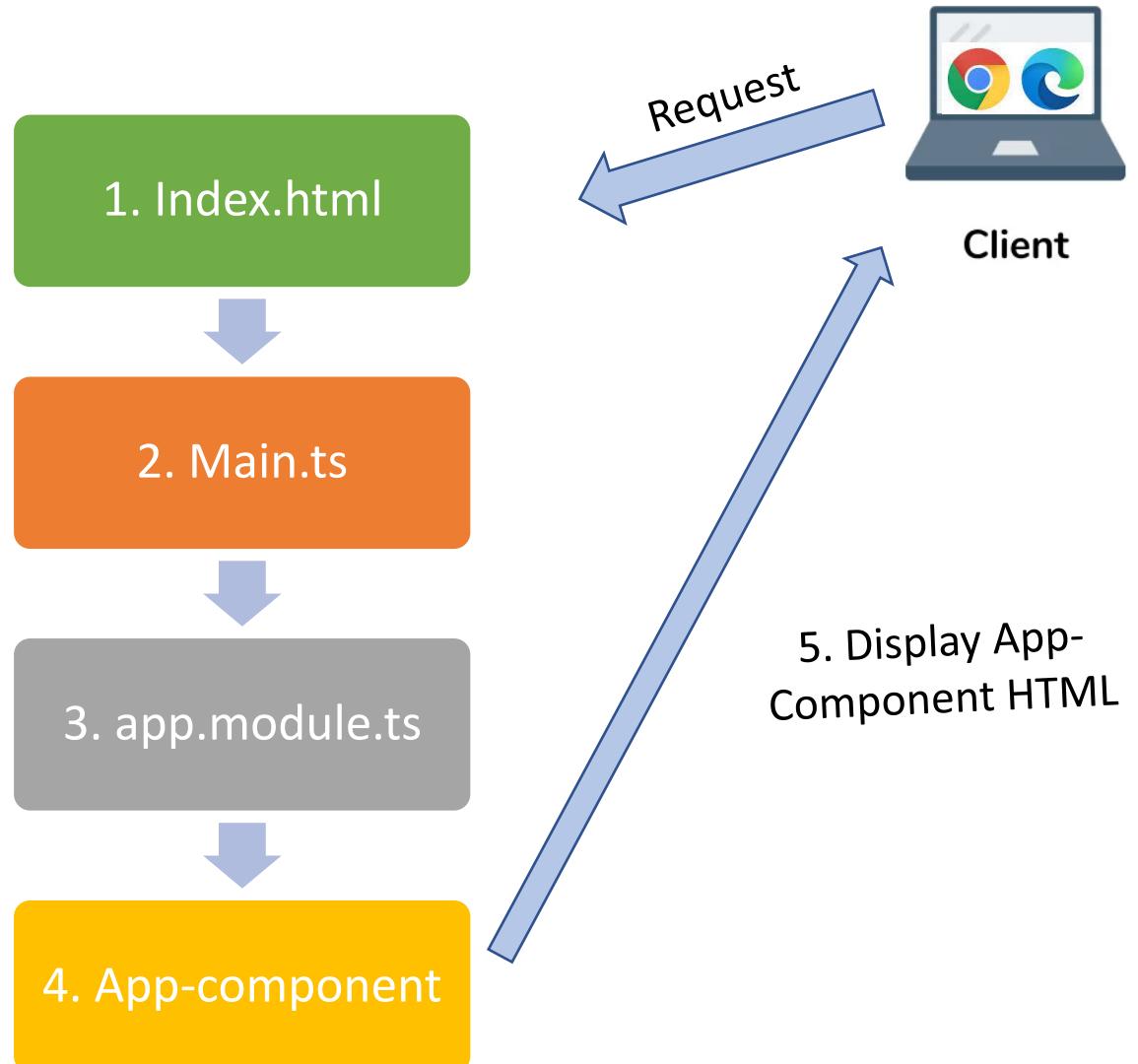
- ❖ Module is a place where you can group the components, directives, pipes, and services, which are related to the application.

For example, here in this module, we have one component two imports.

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10    ],
11    imports: [
12      BrowserModule,
13      AppRoutingModule
14    ],
15    providers: [],
16    bootstrap: [AppComponent]
17  })
18  export class AppModule { }
```

- ❖ 5 Steps executed when client send request from browser:

1. First, request hit index.html page.
2. Second, Index.html will invoke main.js file(which is the javascript version of main.ts file.)
3. Third, main.ts compiles the web-app and bootstraps the app.module.ts file.
4. Fourth, App-Module file will then bootstrap the app-component.
5. Finally, the App-component html will be displayed.

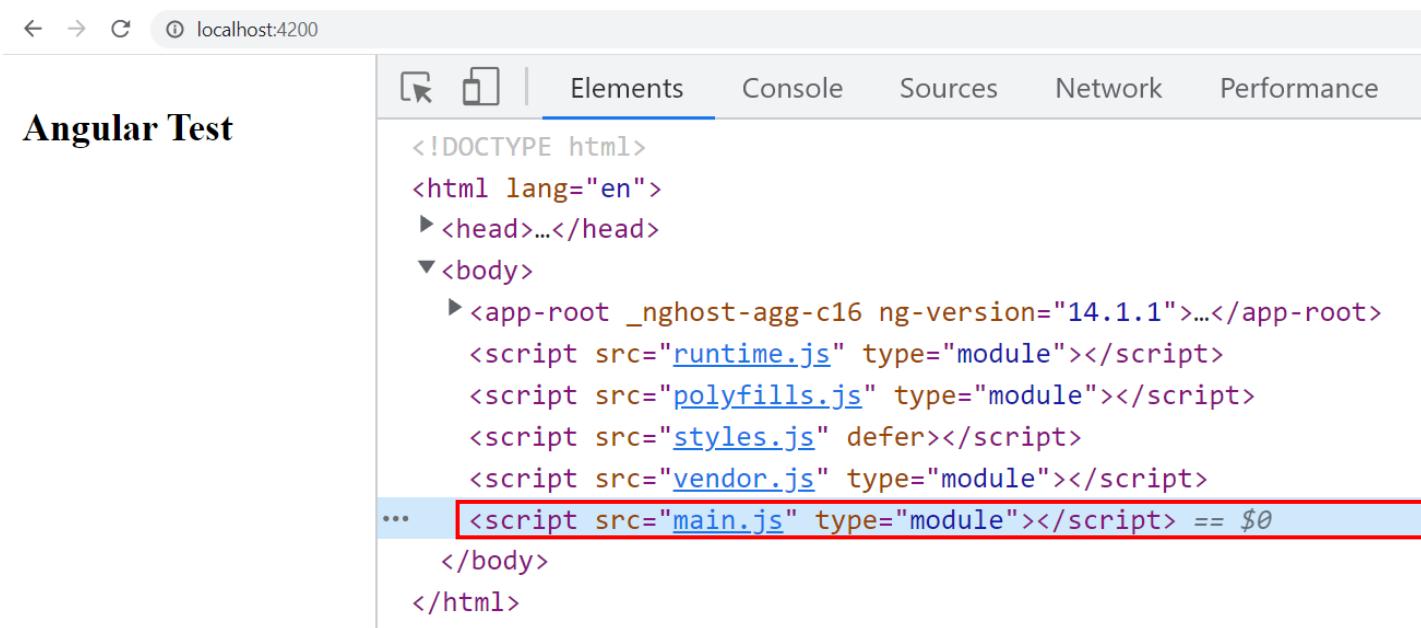


Now let's explore each step-in detail:

1. First step is request will hit index.html.
2. Then second step is, Index.html will invoke main.js file which is the javascript version of main.ts file.

Angular is used to create Single Page Applications. Index.html file is that single page.

See in image, when client hit index.html, then angular cli load these js files, with the html file. One js file among them is main.js, which will be executed then.



The screenshot shows the browser developer tools with the 'Elements' tab selected. The page title is 'Angular Test'. The HTML structure is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <app-root _nghost-agg-c16 ng-version="14.1.1">...</app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    ... <script src="main.js" type="module"></script> == $0
  </body>
</html>
```

The script tag with the src attribute set to 'main.js' is highlighted with a red box.

[Back to chapter index](#)

3. Third step is, main.ts compiles the web-app and bootstraps the app.module.ts file.

In the image, you can see how main.ts has bootstrapped the AppModule file.

```
TS main.ts A X
src > TS main.ts > ...
1  import { enableProdMode } from '@angular/core';
2  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4  import { AppModule } from './app/app.module';
5  import { environment } from './environments/environment';
6
7  if (environment.production) {
8    enableProdMode();
9  }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.error(err));
13
```

4. Fourth step is, App-Module file will then bootstrap the app-component.

In the image, you can see how app.module.ts has bootstrapped the AppComponent file.

5. Fifth step is, App-component html will be sent to client and displayed in the browser.

```
ts app.module.ts A X
src > app > ts app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19 |
```

- ❖ The module which loaded first, when the Angular application start is called Bootstrapped module.
- ❖ By default, AppModule is the Bootstrapped module in Angular project.

```
TS main.ts A X

src > TS main.ts > ...
1  import { enableProdMode } from '@angular/core';
2  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4  import { AppModule } from './app/app.module';
5  import { environment } from './environments/environment';
6
7  if (environment.production) {
8    enableProdMode();
9  }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.error(err));
13
```

- ❖ The component which loaded first, after the module is called Bootstrapped component.
- ❖ By default, AppComponent is the Bootstrapped component in Angular project.

```
ts app.module.ts A X
src > app > ts app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19 |
```

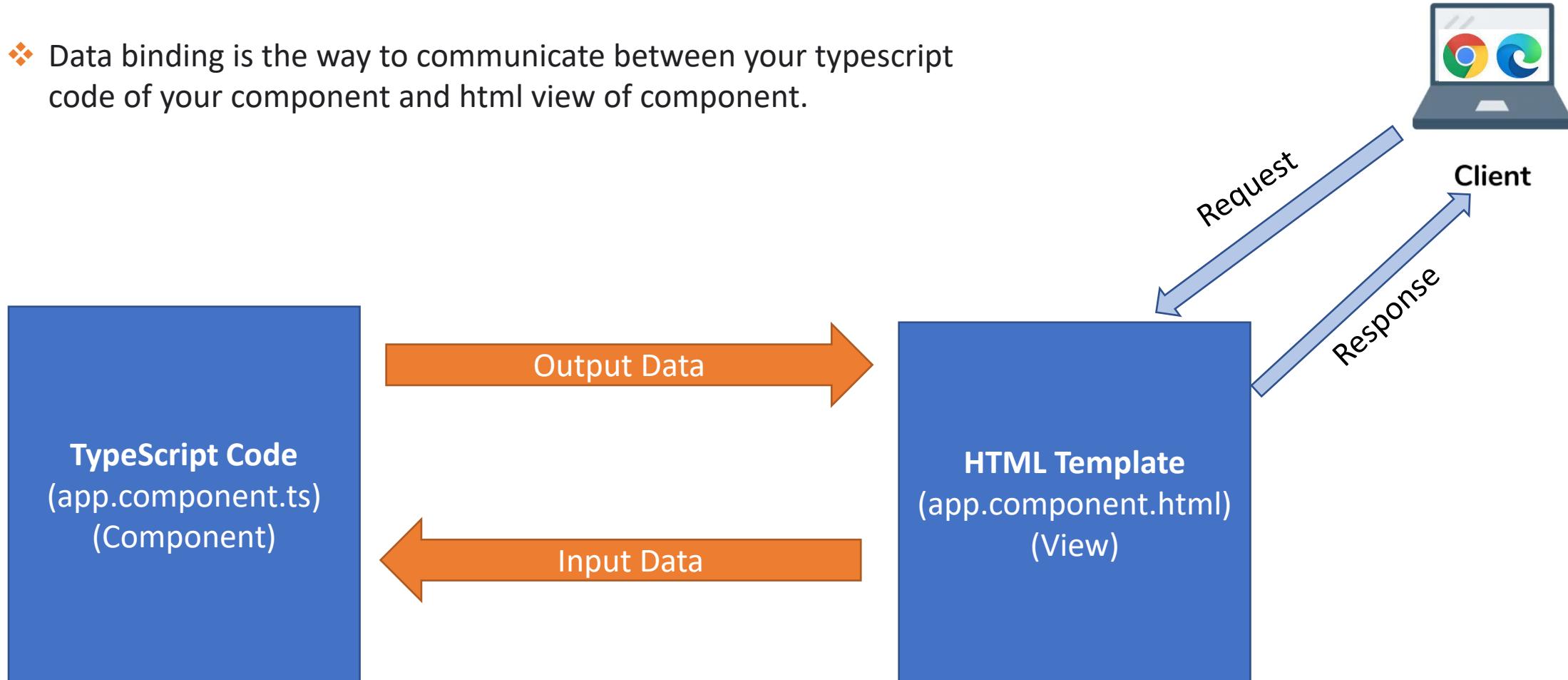


TOP 100 ANGULAR INTERVIEW QUESTIONS

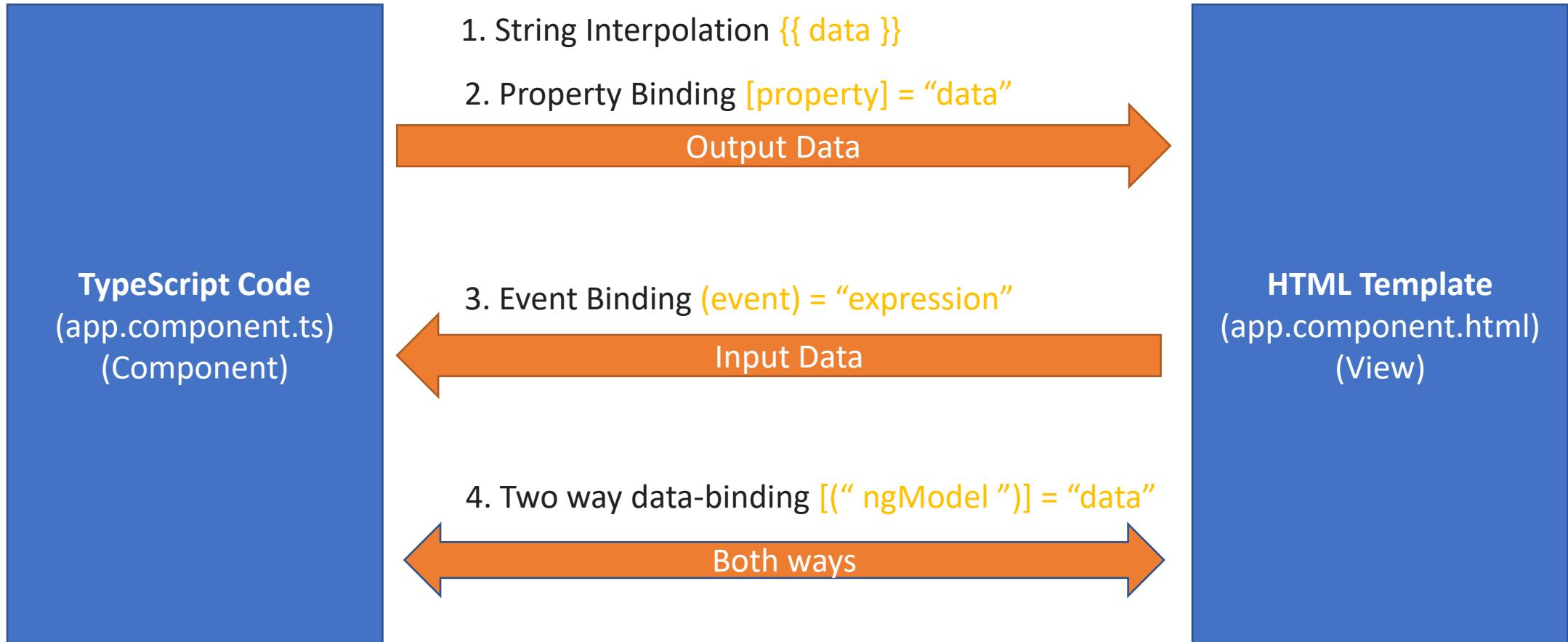
Chapter 3: Data Binding

- Q15. What is **Data Binding** in Angular?
- Q16. What is **String Interpolation** in Angular?
- Q17. What is **Property Binding** in Angular?
- Q18. What is **Event Binding** in Angular?
- Q19. What is **Two way Binding** in Angular?

- ❖ Data binding is the way to communicate between your typescript code of your component and html view of component.



- ❖ 4 Types of data binding in Angular:



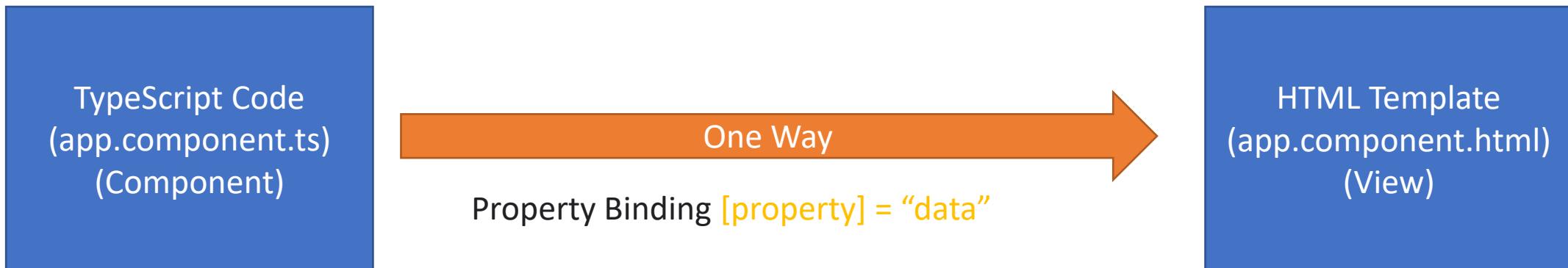
- ❖ String Interpolation is a one-way data-binding technique that is used to transfer the data from a TypeScript code(component) to an HTML template (view).



1. String interpolation can work on string type only not numbers or any other type.
2. It is represented inside {{data}} double curly braces.

- ❖ Property binding is a superset of interpolation.

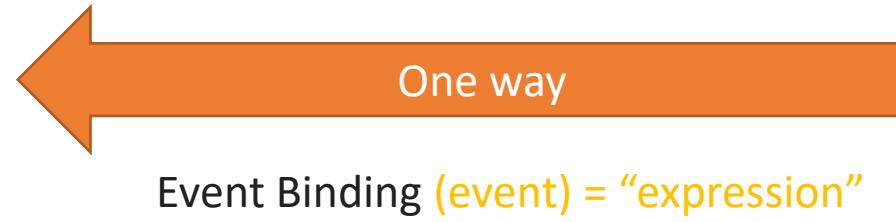
It can do whatever interpolation can do. In addition, it can set an element property to a non-string data value like Boolean.



- ❖ When to use property binding and when to use interpolation?
 1. When you want to transfer string only, then use interpolation because of its **simplicity**(just {{data}}) to use.
 2. When you want to transfer any type other than string, then use property binding.

- ❖ Event binding is used to handle the events raised by the user actions like button click.

TypeScript Code
(app.component.ts)
(Component)



HTML Template
(app.component.html)
(View)

- ❖ Two-way data binding in Angular will help users to exchange data from the view to component and then from component to the view at the same time.



TypeScript Code
(app.component.ts)
(Component)



Two way data-binding [“ ngModel ”] = “data”

HTML Template
(app.component.html)
(View)



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 4: Directives

Q20. What are Directives? What are the type of directives?

Q21. What is *ngIf Structural directive?

Q22. What is *ngFor Structural directive?

Q23. What is *ngSwitch Structural directive?

Q24. What is [ngStyle] Attribute directive?

Q25. What is [ngClass] Attribute directive?

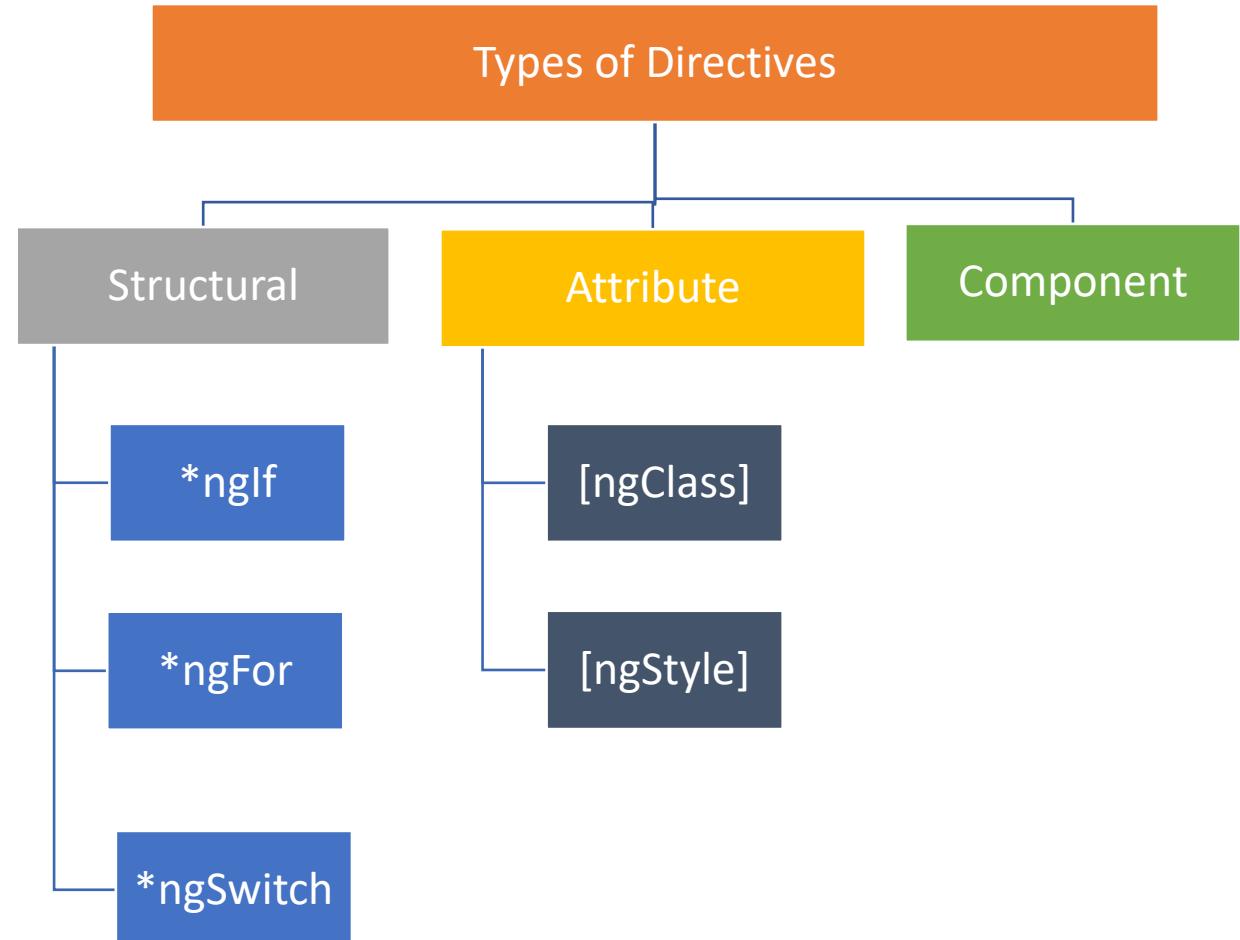
Q26. What is the difference between Component, Attribute and Structural Directives?

- ❖ Directives are classes that add additional behavior to elements in your Angular applications.
- ❖ *For example, in below image, we can change the color of button element by using directives.*

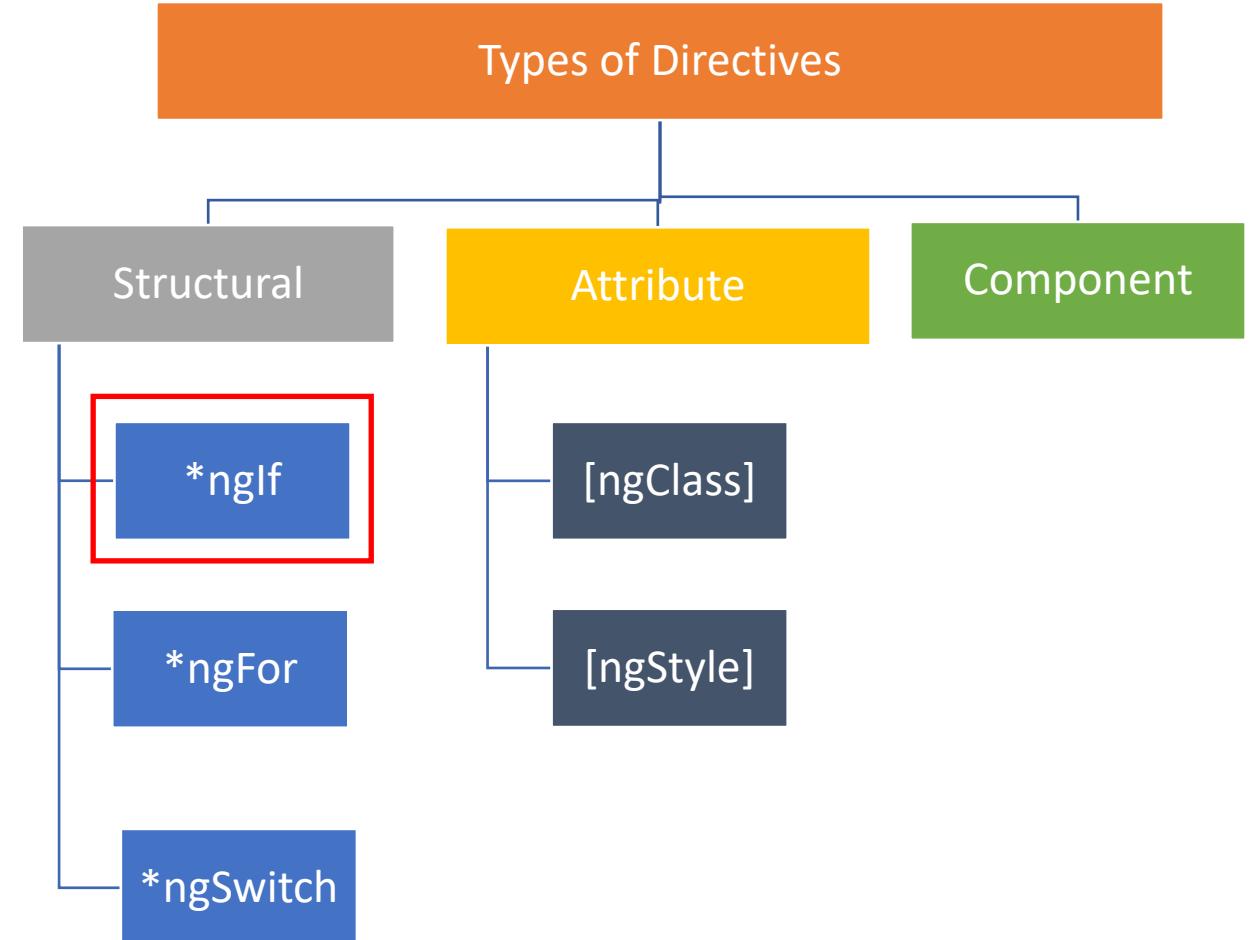


❖ Types of Directives:

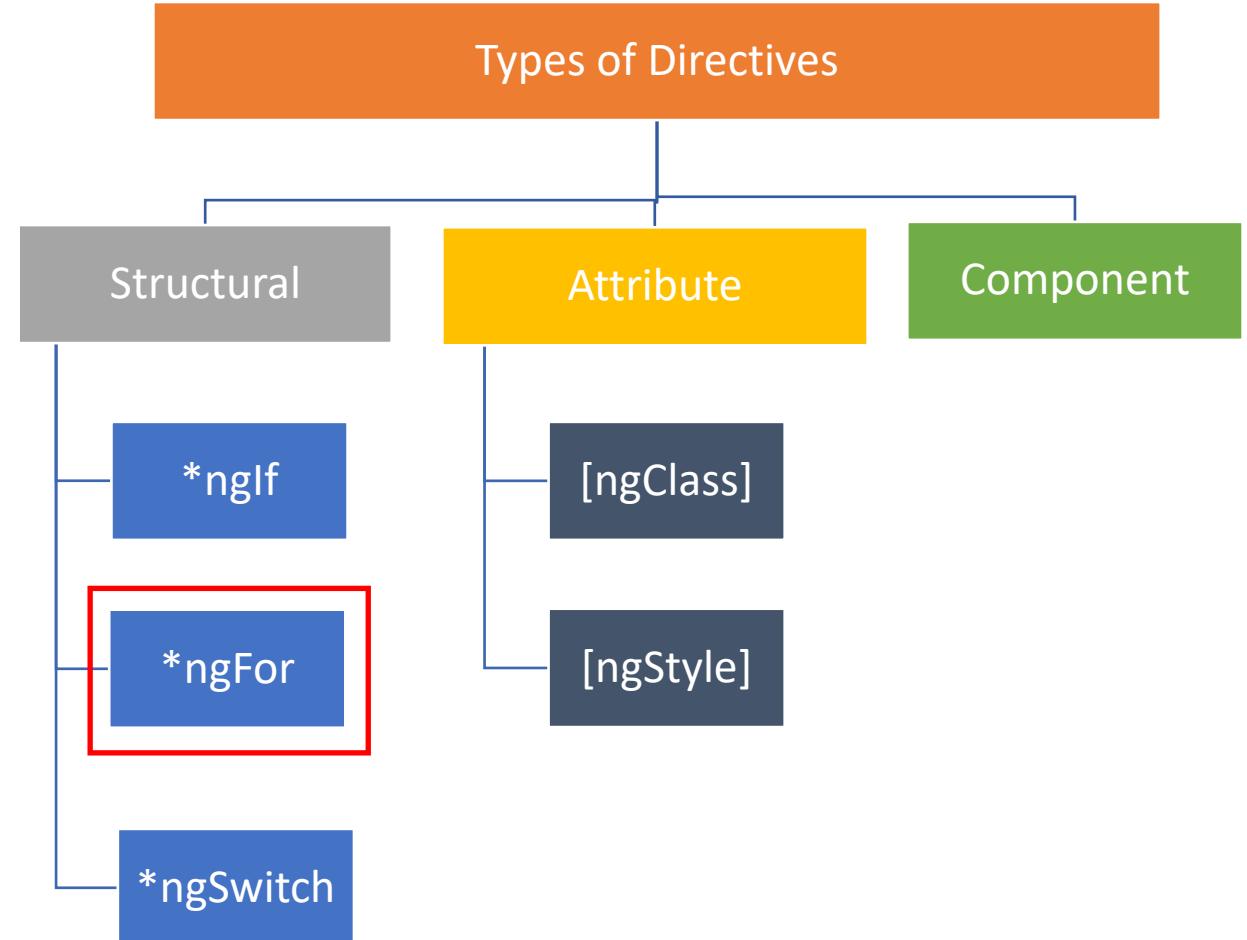
1. **Structural Directives** are classes that change appearance of DOM by adding or removing elements.
2. **Attribute directives** change the appearance or behavior of an element.
3. **Component directive** are directives with its own template.



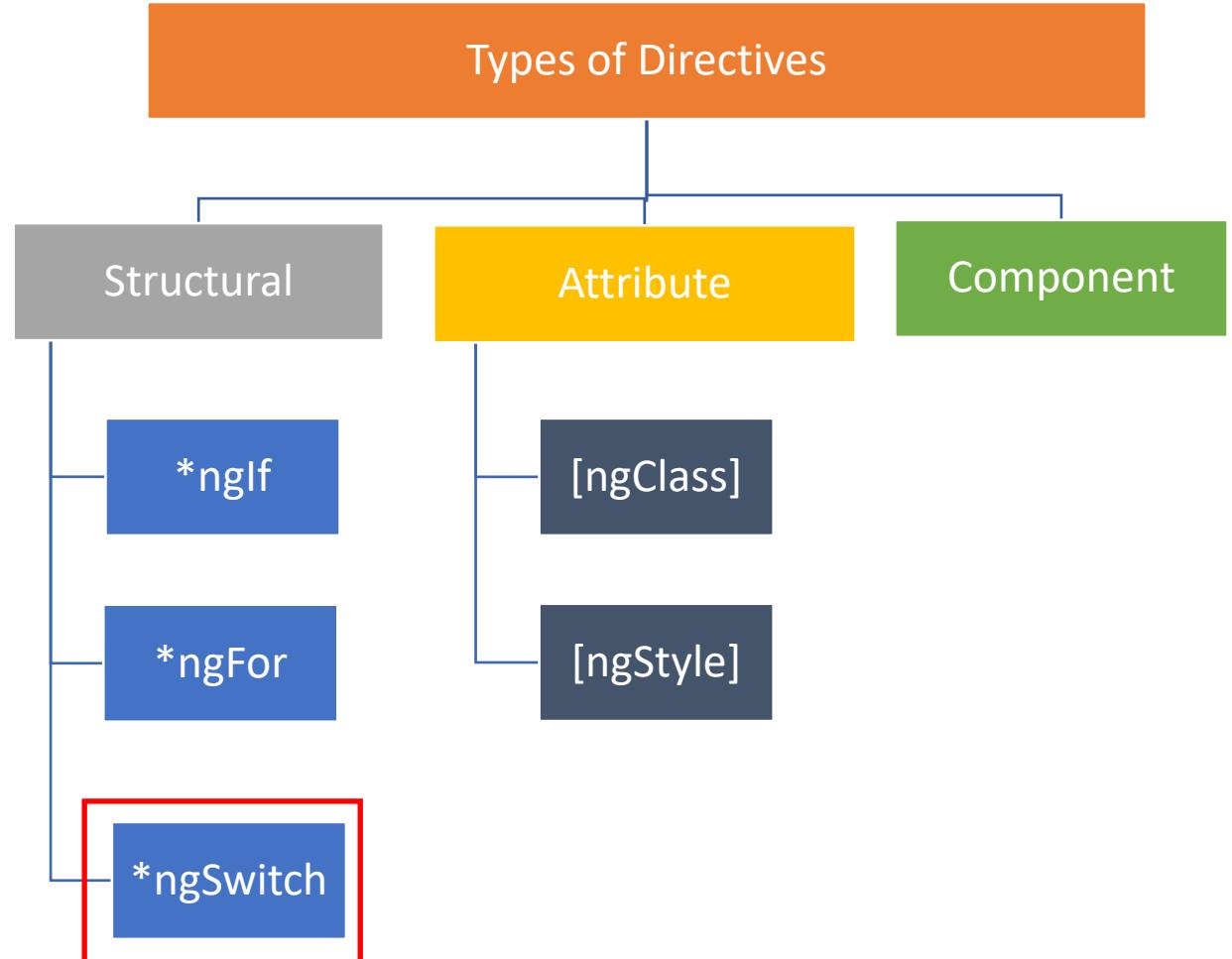
- ❖ *ngIf is a structural directive, which is used to add or remove items based on the **if condition**.



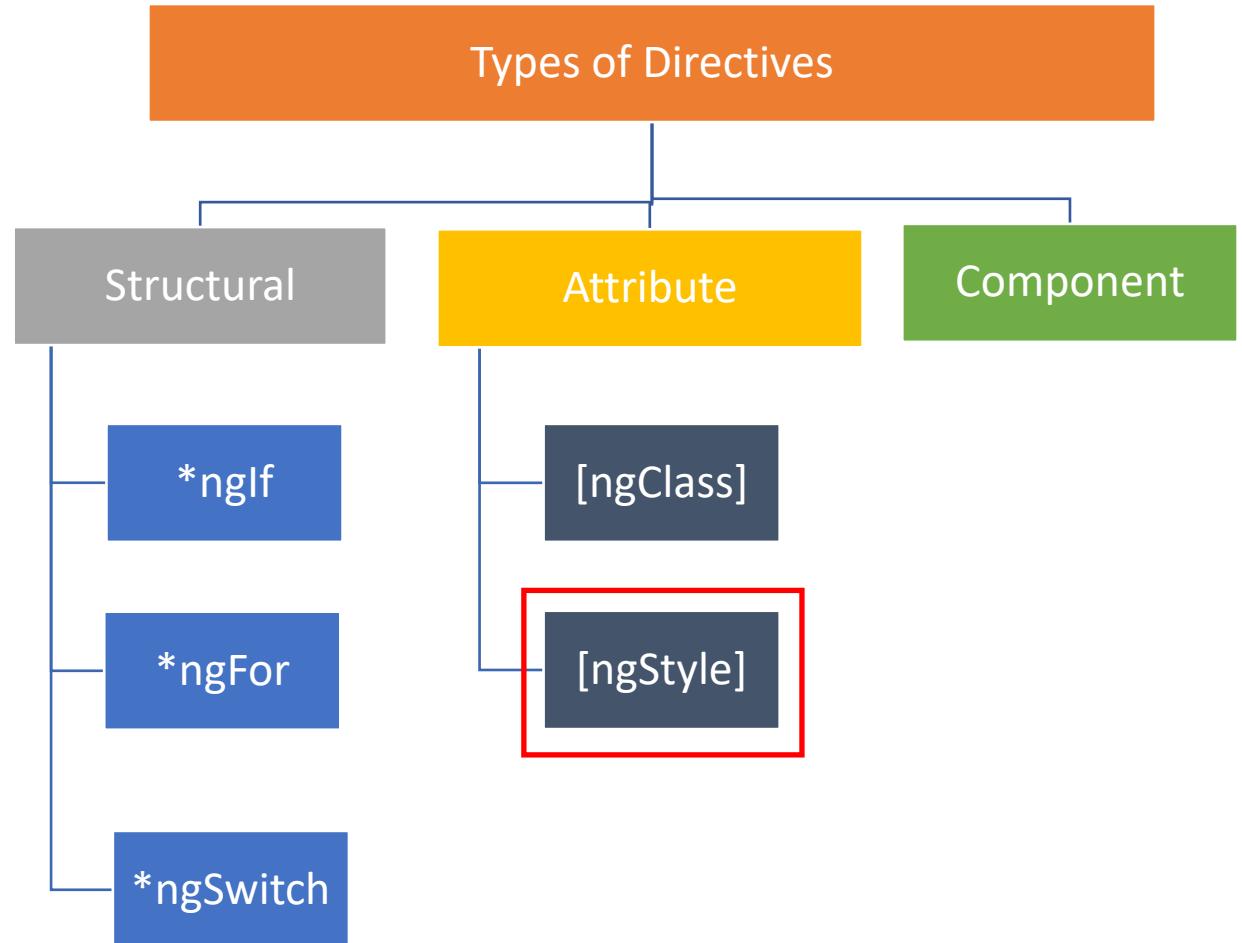
- ❖ *ngFor is a structural directive, which is used to **iterate** a list of items and then display them.



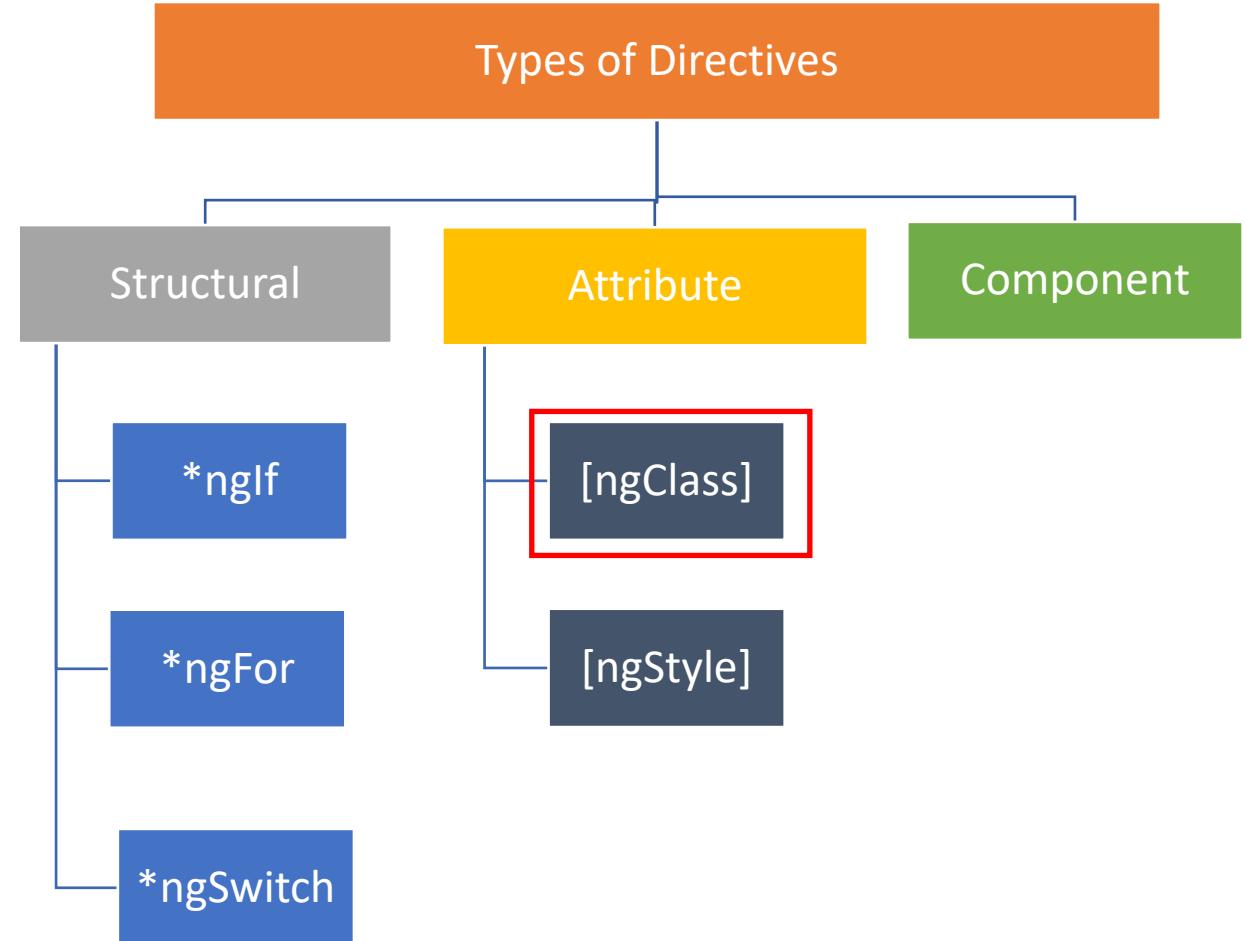
- ❖ ngSwitch is a structural directive, which is used in combination with *ngSwitchCase and *ngSwitchDefault that are both structural directives.



- ❖ [ngStyle] is an attribute directive, which is used to update **styles** of the HTML element.



- ❖ [ngClass] is an attribute directive, which is used to add and remove **CSS classes** on an HTML element.



Component Directive	Structural Directive	Attribute Directive
1. Component directive is responsible for showing the first whole view. It is the most used one.	Structural directive is responsible for adding and deleting html elements in the view.	Attribute directive is responsible for changing the appearance of view by adding or removing styles/cssclasses of the html elements.
2. Starts with @ sign. Like: @Component	Starts with * sign. Like: *ngIf, *ngFor, *ngSwitch	Set inside square brackets. Like: [ngClass], [ngStyle]



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 5: Decorator & Pipes

Q27. What is **Decorator**?

Q28. What are the **types of Decorator**?

Q29. What are **Pipes**? What are the types of Pipes & Parameterized Pipes?

Q30. What is **Chaining Pipes**?

- ❖ Angular decorators store **metadata** about a class, method, or property.

For example, in image @Component decorator store metadata information about the app-component.

- ❖ All decorators are represented with @ symbol.

- ❖ What is Metadata?

Metadata is "data that provides information about other data".

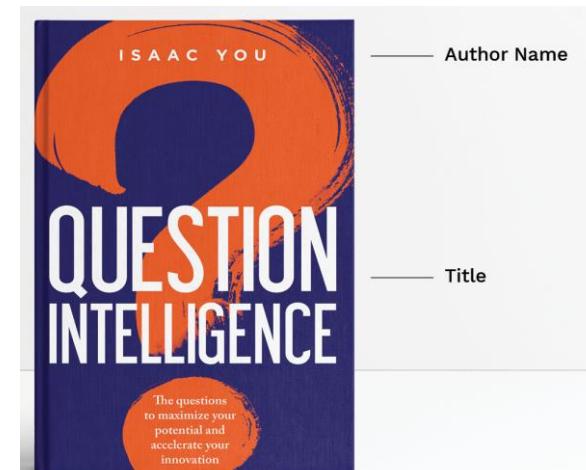
For example, in image we have a book. Now things like author name, title, price, publisher etc etc, all this information about the book is metadata only.

Meaning metadata gives you some information about the book, but it is not the main content or data of the book. This is metadata.

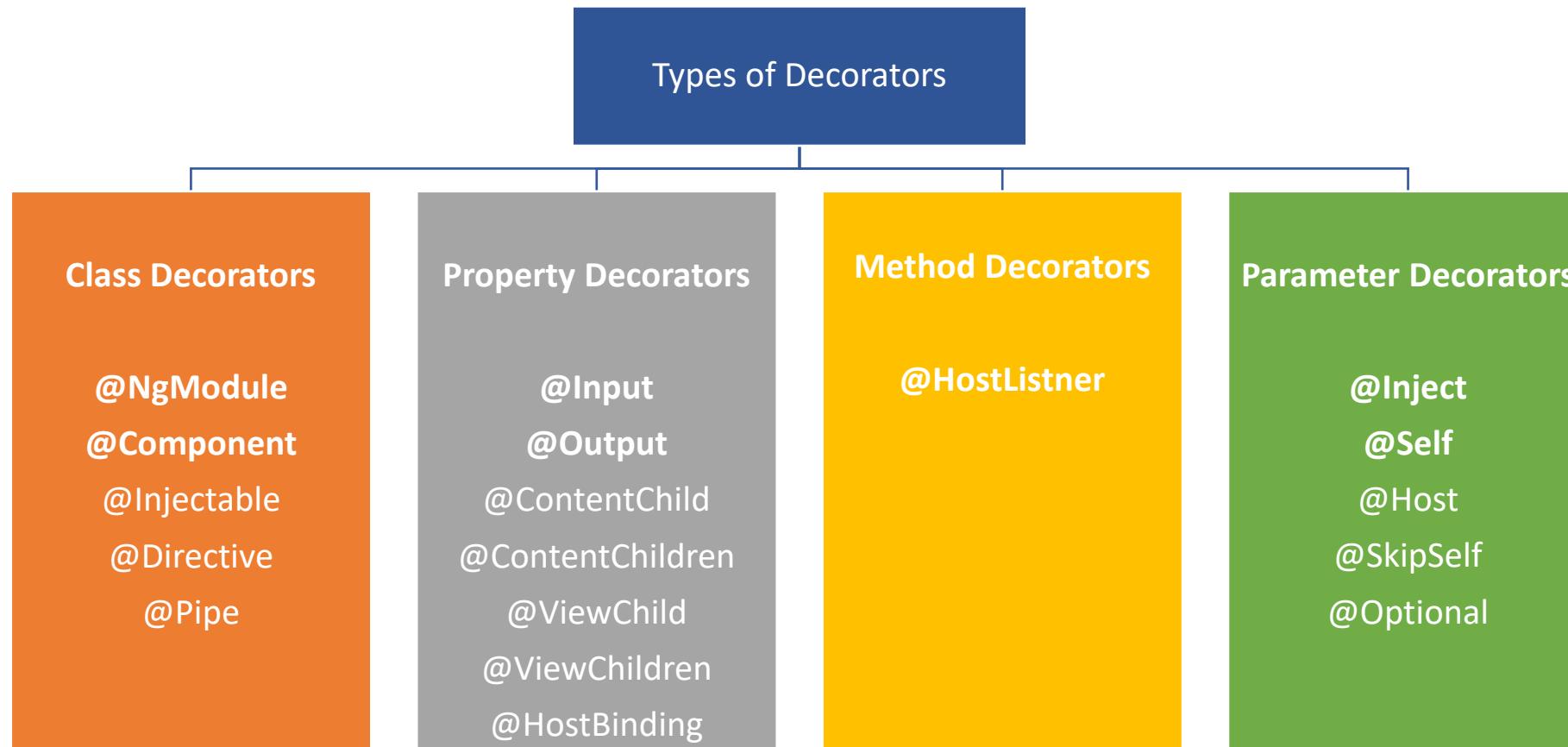
```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'first-interview';
}
```

Metadata



- ❖ Here are the Types of decorators – Class, Property, Method and Parameter are the categories of decorators. We will use these decorators later in different parts of the Angular applications.



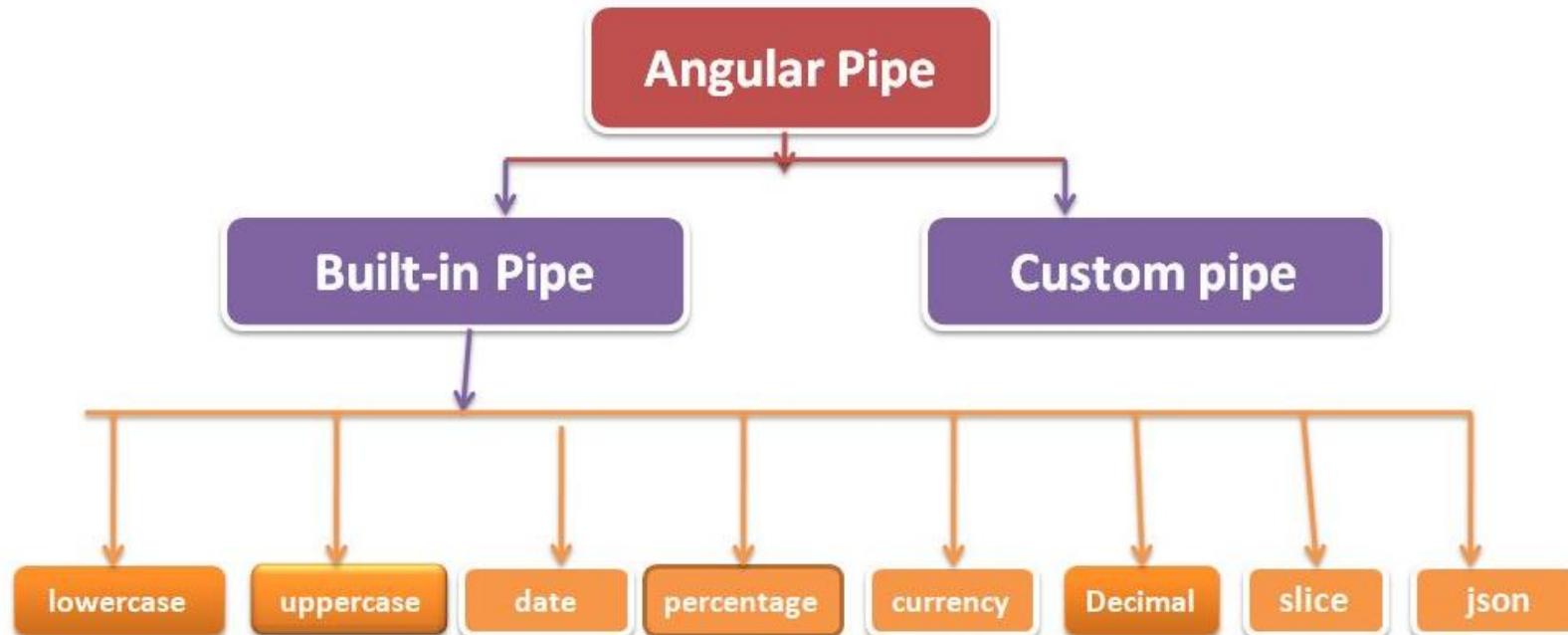
- ❖ Pipes are simple functions which accept an input value and return a **transformed value**.

For example, in image the input is “Interview Happy” and the transformed value is “interview happy”.

```
<h2>{{"Interview Happy" | lowercase}}</h2>
<!-- Output - interview happy --&gt;
&lt;h2&gt;{{"Interview Happy" | uppercase}}&lt;/h2&gt;
<!-- Output - INTERVIEW HAPPY --&gt;</pre>
```

❖ Types of Pipes:

1. Built-in Pipes are provided by Angular framework. Like lowercase, uppercase, date etc.
2. Custom pipe, we can create and develop ourselves.



- ❖ When we pass any parameters to the pipes, it is called **parameterized pipes**.

```
<!-- Slice pipe will trim 10 characters from start. -->
<!-- We are passing 10 as the parameter. -->
<h2>{{"Interview Happy" | slice:10}}</h2>
<!-- Output --> Happy
```

- ❖ The chaining pipes use multiple pipes on a data input.

For example, in below image one pipe is for slice and then other pipe will further transform the result to uppercase

```
<!-- Chaining Pipes -->

<h2>{{"Interview Happy" | slice:10 | uppercase}}</h2>

<!-- Output - HAPPY -->
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

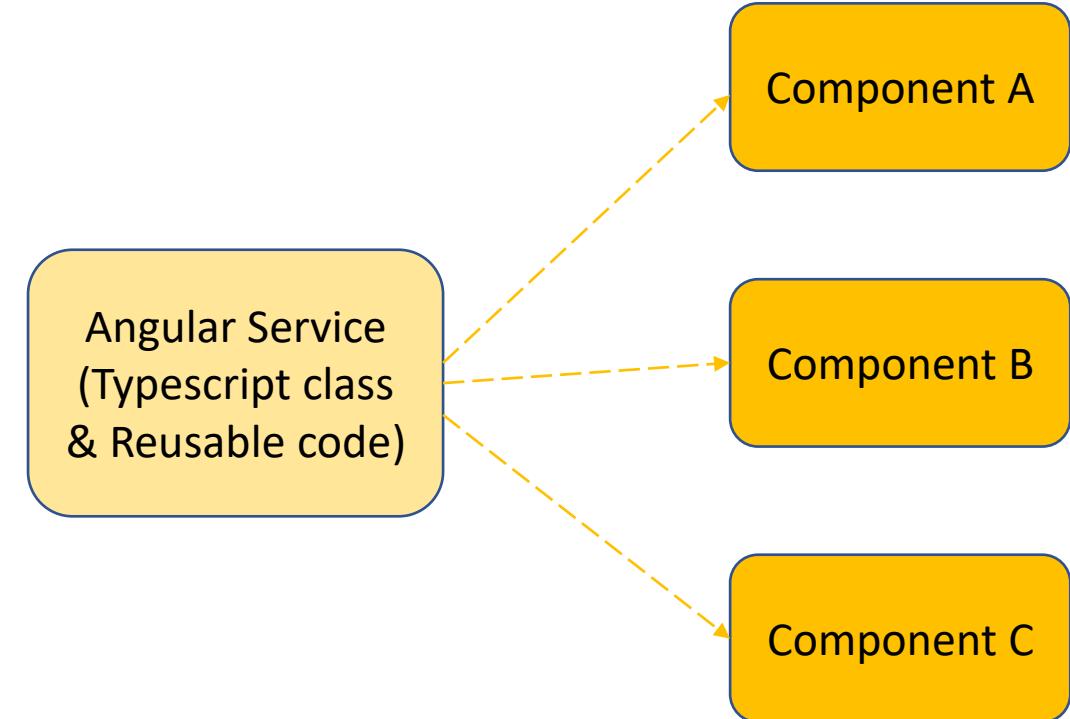
Chapter 6: Services & Dependency Injection

- Q31. Explain **Services** with Example?
- Q32. How to **create Service** in Angular?
- Q33. How to use **Dependency Injector** with Services in Angular?
- Q34. What is Hierarchical **Dependency Injection**?
- Q35. What is **Provider** in Angular?
- Q36. What is the role of **@Injectable** Decorator in a Service?

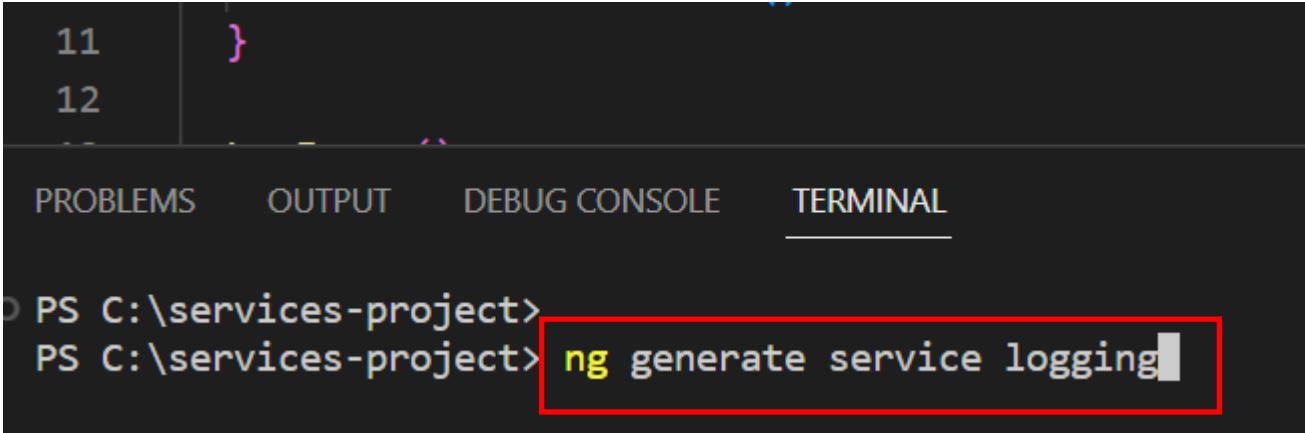
- ❖ A service is a typescript class and **reusable code** which can be used in multiple components.

- ❖ How to use services in components?

Service can be implemented in components with the help of **dependency injection**.



- ❖ Below is the command to generate service of name “logging” in Angular



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\services-project> PS C:\services-project> ng generate service logging
```

- ❖ 4 Steps to inject service dependency in the component:

1. First step is, create the service – (LoggingService).

```
//1. Create the Service
@Injectable()

export class LoggingService{
  logToConsole()
  {
    console.log("clicked logged");
  }
}
```

2. Second step is, set the providers as the service name(LoggingService), inside the any component decorator as shown below.

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  providers: [LoggingService]
})

export class LoginComponent{}
```

3. Third step is, inside the **constructor** parameter create a new property(loggingService) and then assign the Service type(LoggingService) here.

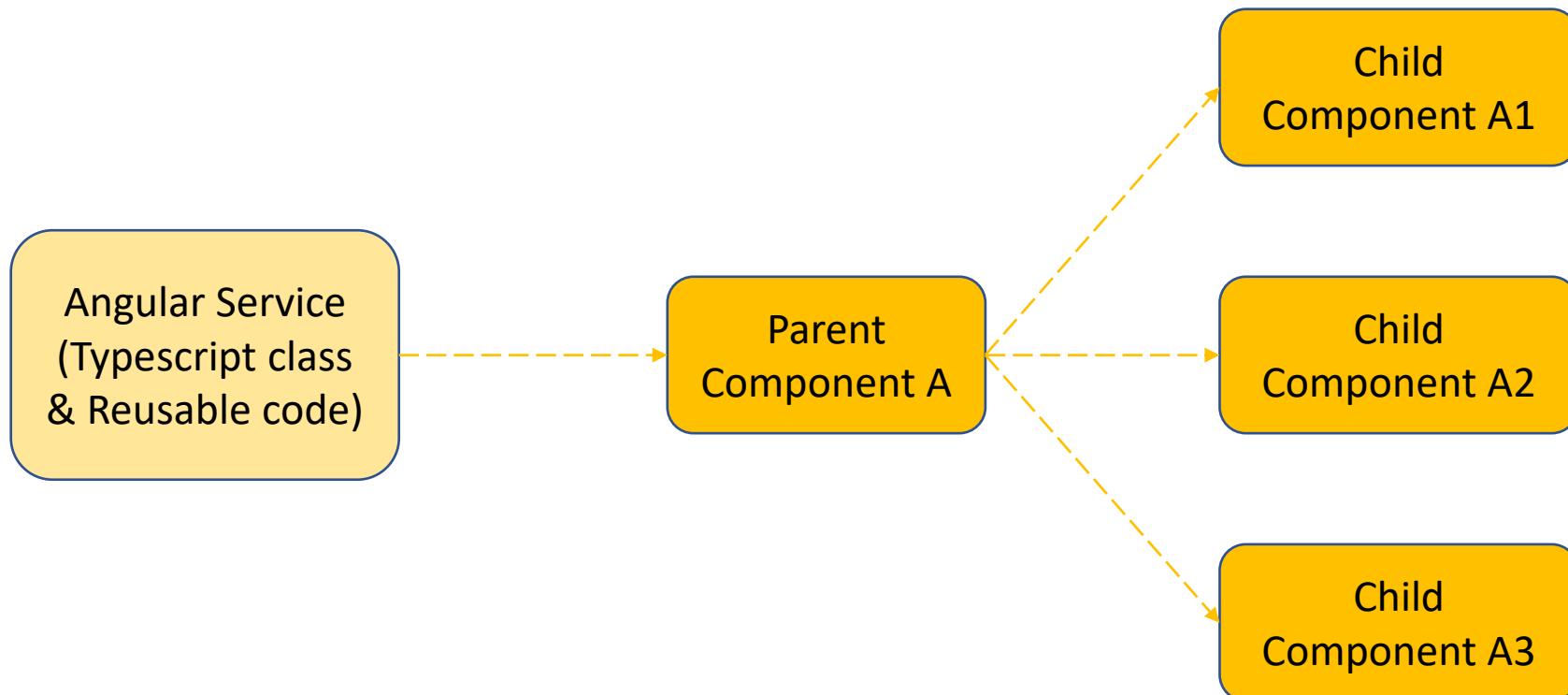
```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  providers: [LoggingService]
})

export class LoginComponent{
  constructor(private loggingService: LoggingService){
  }
}
```

4. Finally use the property(loggingService), to call the method(logToConsole) of the LoggingService.

```
export class LoginComponent{  
  constructor(private loggingService: LoggingService){  
  }  
  onClick(){  
    this.loggingService.logToConsole();  
  }  
}
```

- ❖ As per Angular, if we will inject the service in parent component, then service will be available to all it's child components, but if we will inject the service directly in child component then it will not be available for Parent and siblings components.



- ❖ A provider is an object declared inside decorators which inform Angular that a particular service is available for injecting inside the components.

Provider can be set in module or in the component as shown below.

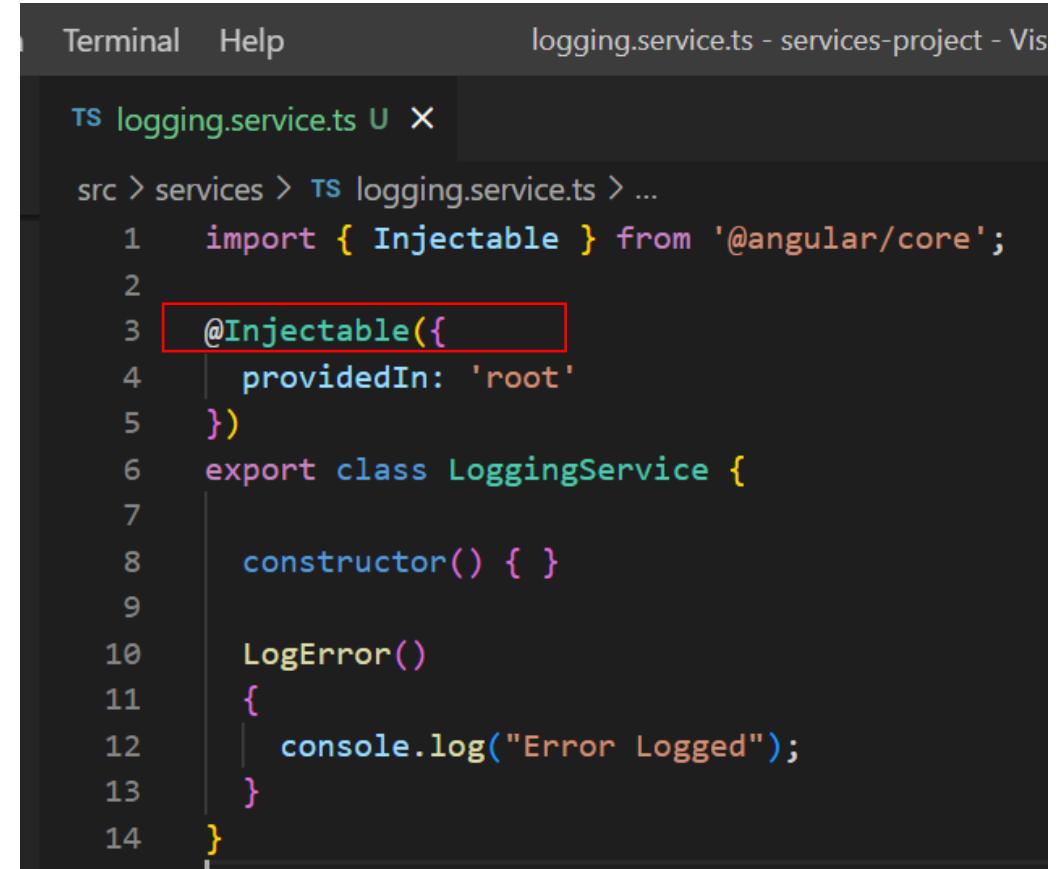
```
10  @NgModule({
11    declarations: [
12      AppComponent,
13      LoginComponent,
14      MenuComponent
15    ],
16    imports: [
17      BrowserModule,
18      AppRoutingModule
19    ],
20    providers: [LoggingService],
21    bootstrap: [AppComponent, LoginComponent]
22  })
23  export class AppModule { }
24  |
```

```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'],
  providers: [LoggingService]
})
export class LoginComponent implements OnInit {

  constructor(private loggingService: LoggingService) {
    this.loggingService.LogError();
    //const ls = new LoggingService();
    //ls.LogError();
    // console.log("Error Logged1");
    // console.log("Error Logged2");
    // console.log("Error Logged3");
  }
}
```

- ❖ With `@Injectable` decorator one service can be used by another service.

For example, in image we set the `LoggingService` as `Injectable`. That means, now any other service also can use `LoggingService` methods.



```
Terminal Help logging.service.ts - services-project - Vis
TS logging.service.ts U X
src > services > TS logging.service.ts > ...
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class LoggingService {
7
8   constructor() { }
9
10  LogError()
11  {
12    console.log("Error Logged");
13  }
14}
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 7: Decorators & Lifecycle-Hooks

Q37. What are Parent-Child Components?

Q38. What is @Input Decorator? How to transfer data from Parent component to Child component?

Q39. What is @Output Decorator and Event Emitter?

Q40. What are Lifecycle Hooks in Angular?

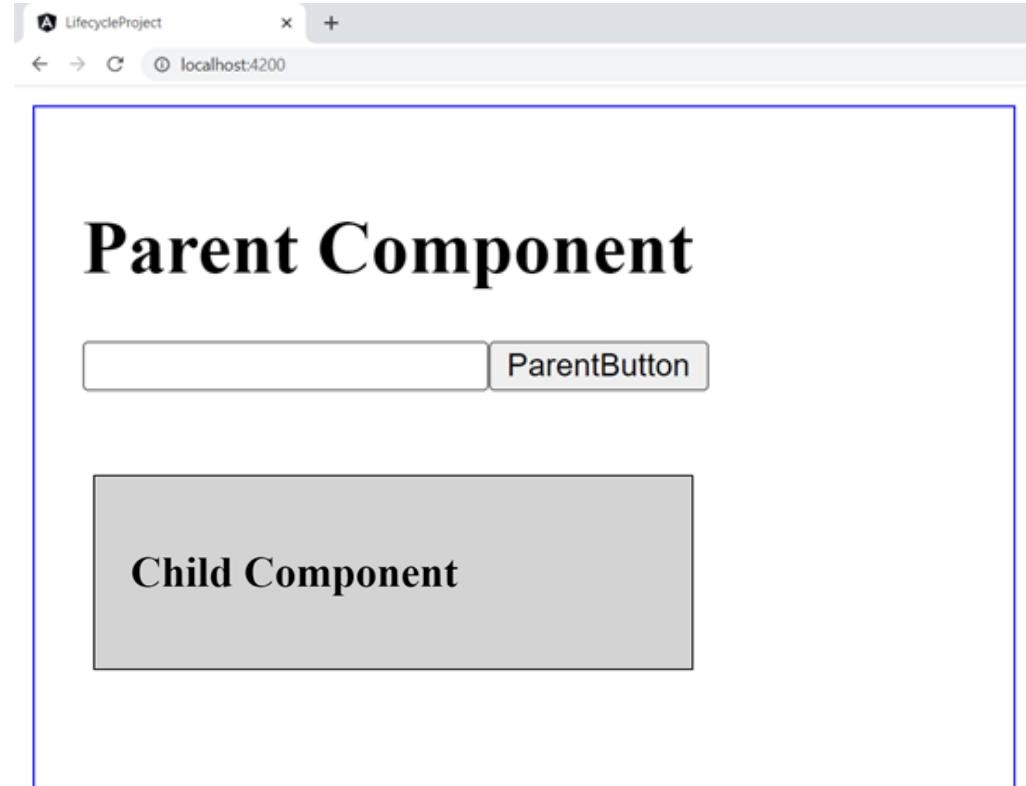
Q41. What is a Constructor in Angular?

Q42. What is ngOnChanges life cycle hook in Angular?

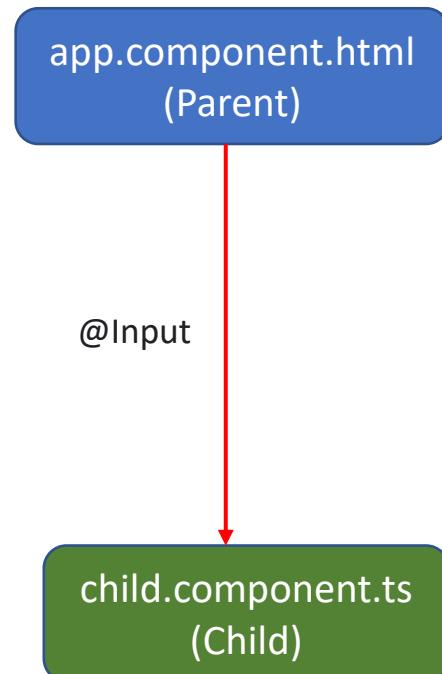
Q43. What is ngOnInit life cycle hook in Angular?

Q44. What is the difference between constructor and ngOnInit?

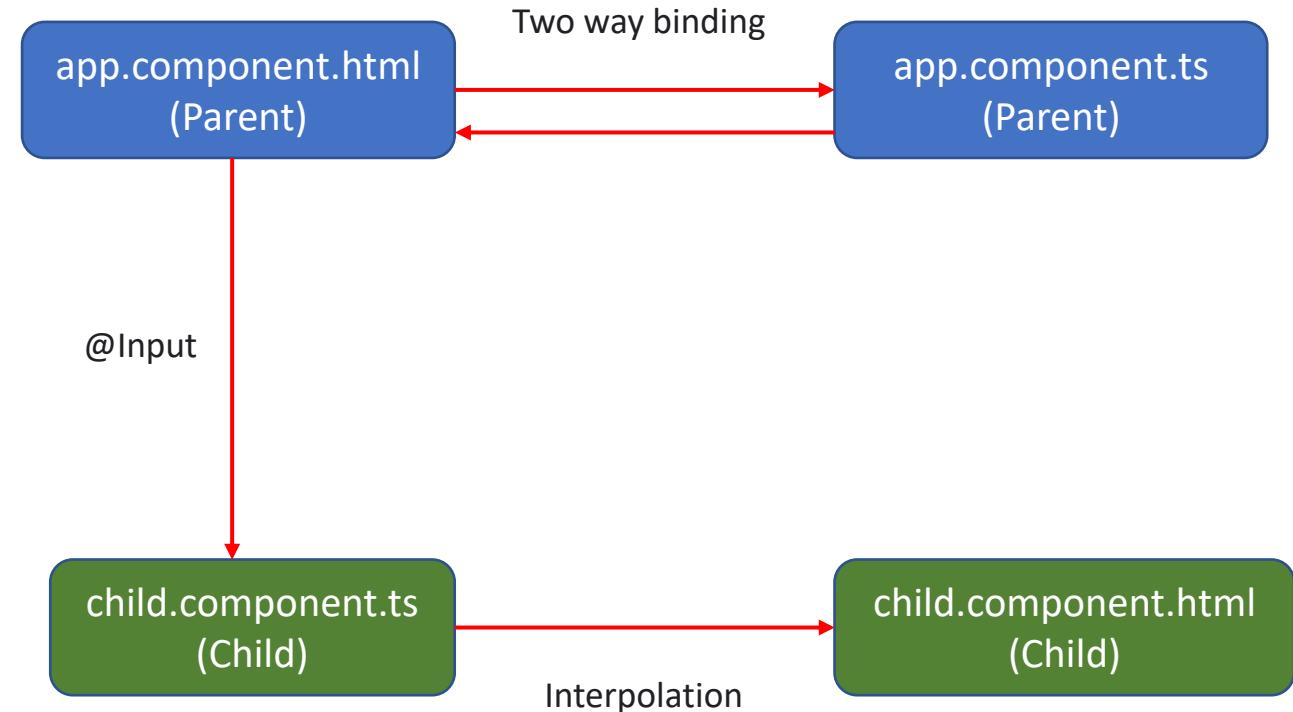
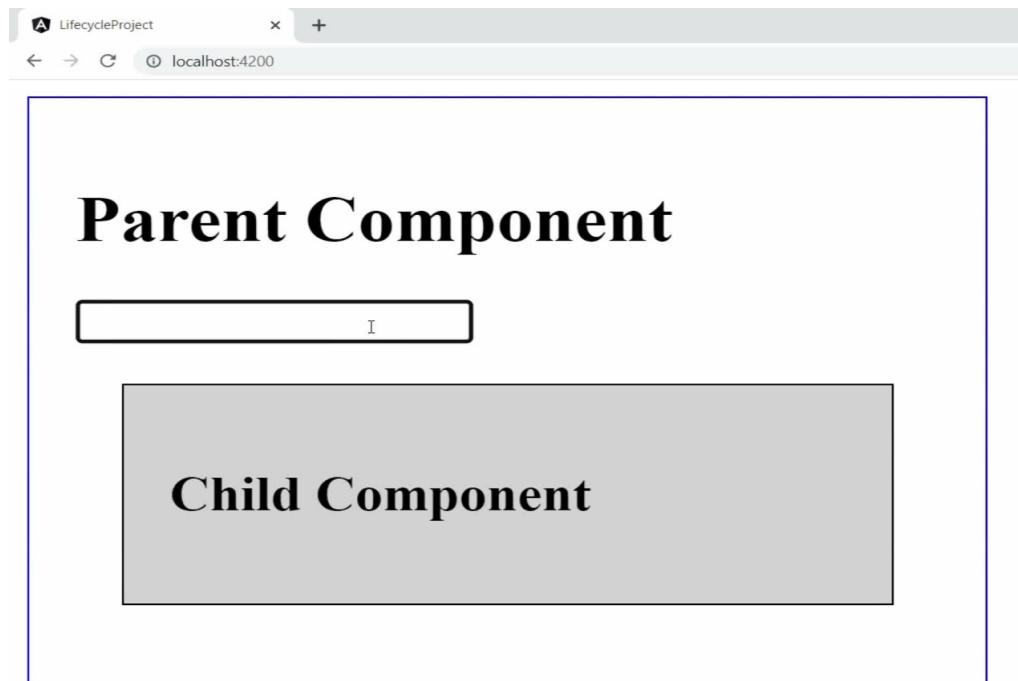
- ❖ Parent component is the outer component.
- ❖ Child components are the components inside the parent component.
- ❖ Most of the time app-component is only the parent component and rest of the components are child components



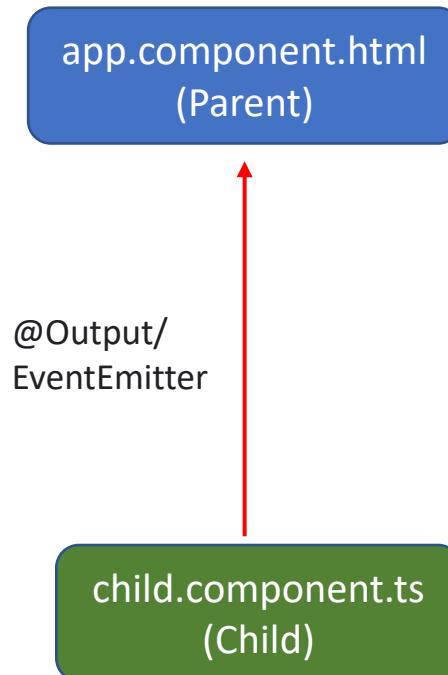
- ❖ @Input decorator is used to pass data from parent component to child component.



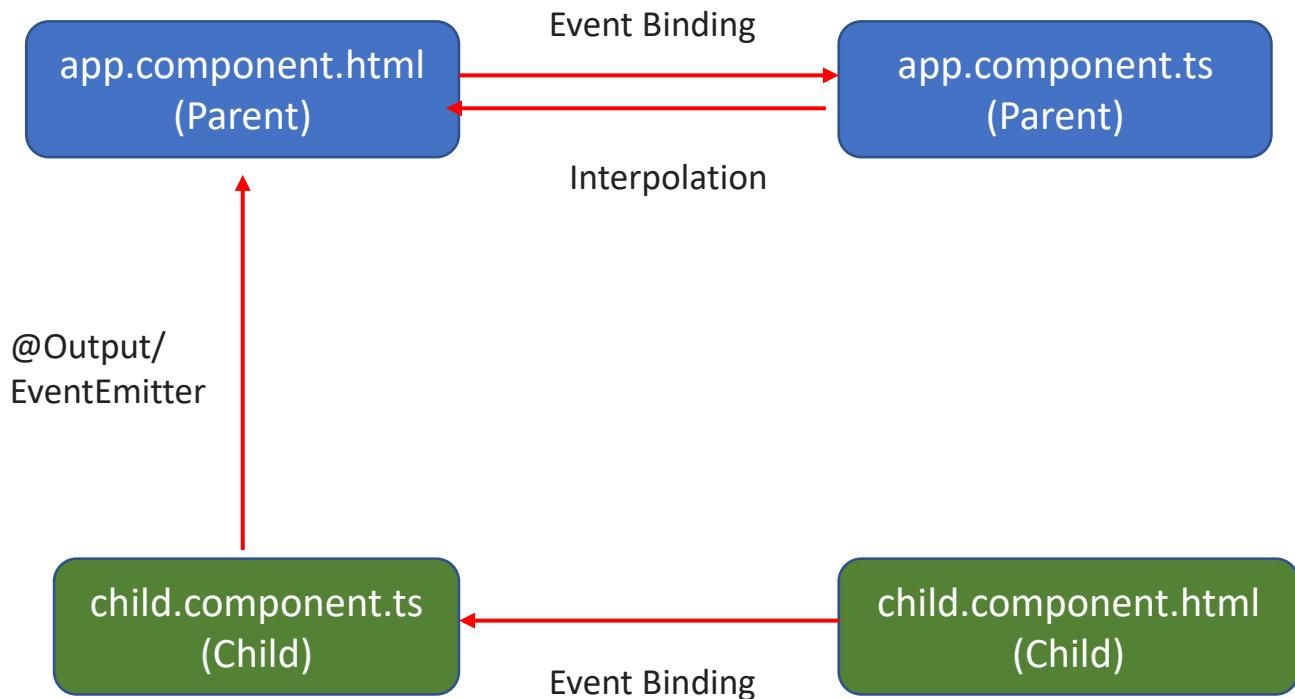
- ❖ @Input decorator is used to pass data from parent component to child component.



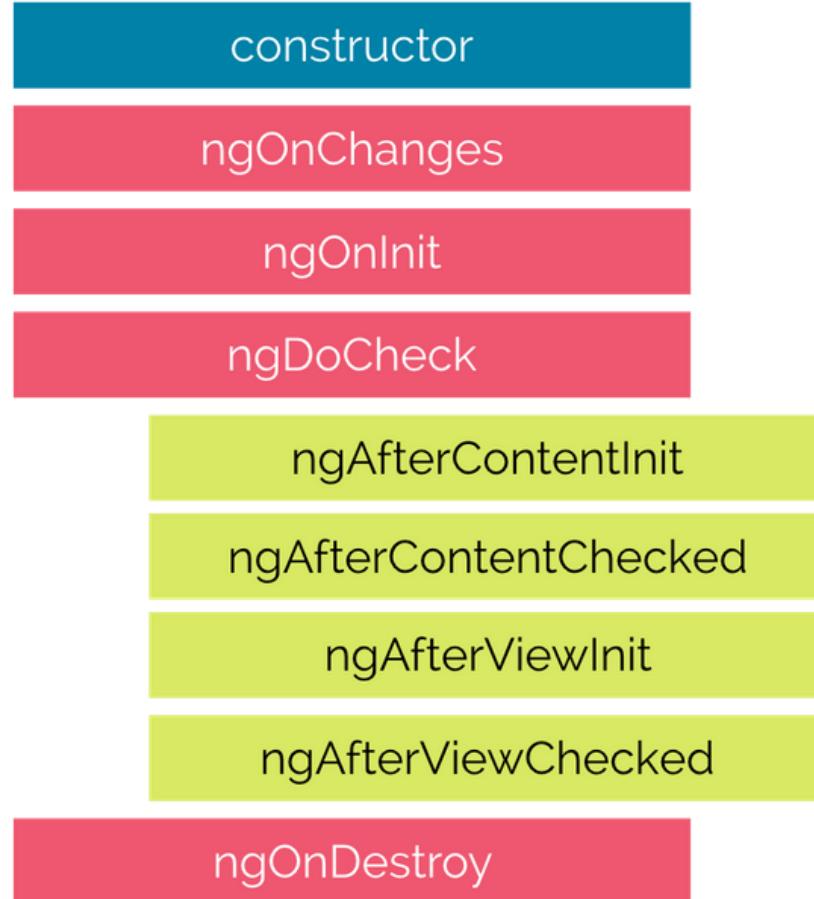
- ❖ @Output decorator and event emitter together are used to pass data from child component to parent component.



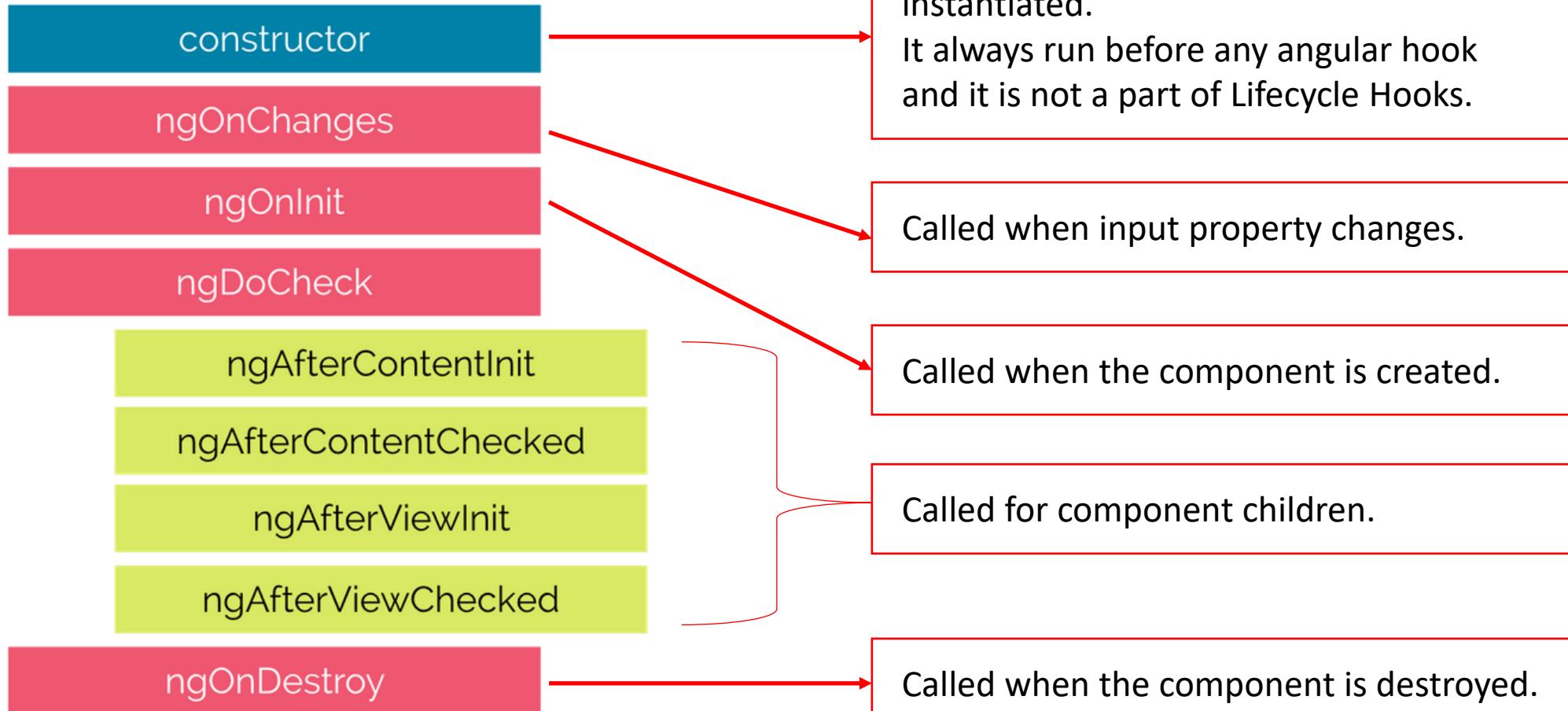
- ❖ @Output decorator and event emitter together are used to pass data from child component to parent component.



- ❖ A component from creation to destruction goes through several stages and these stages are the life cycle hooks.
- ❖ The stages will cover activities like:
 - Component instantiating.
 - Rendering the component html view.
 - Creating the child components(optional).
 - Destroying the component.
- ❖ Angular provides these hooks, inside them we can code for appropriate functionality.



- ❖ Here are some details about the hooks:



- ❖ The constructor is a method in a **TypeScript class** that automatically gets called when the class is being **instantiated**.

- ❖ Whether constructor is a life cycle hook?

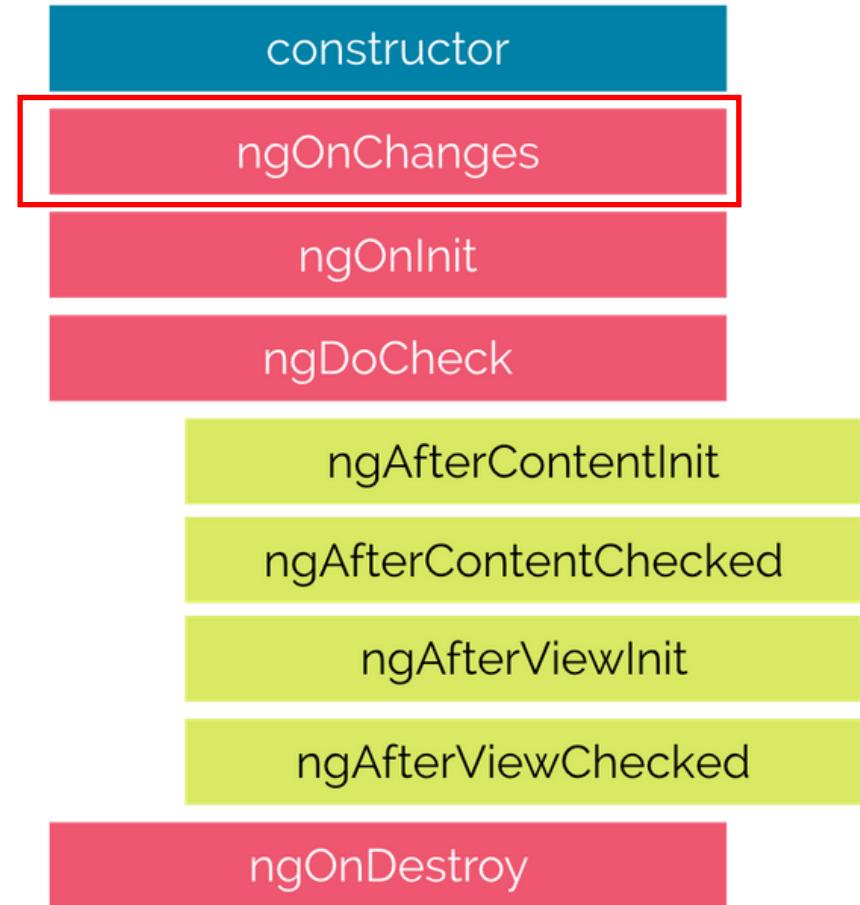
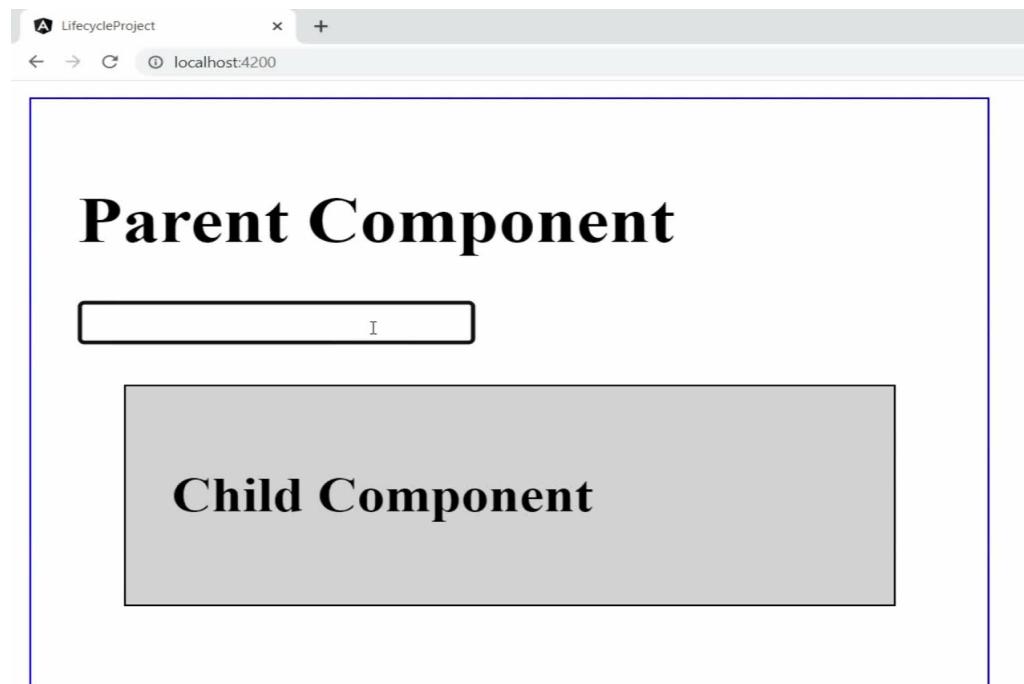
Constructor always run before any angular hook and it is not a part of Lifecycle Hooks.

- ❖ One example, when we have to write code inside constructor?

Constructor is widely used to inject **dependencies(services)** into the component class.

```
ts sample.component.ts U ×
src > app > sample > ts sample.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-sample',
5    templateUrl: './sample.component.html',
6    styleUrls: ['./sample.component.css']
7  })
8  export class SampleComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14
15 }
16
```

- ❖ The ngOnChnages is the first life cycle hook, which angular fires when it detects any changes to the **input** property.



1. OnInit signals the activation of the **created** component.

2. This is the second hook and called after ngOnChanges.

3. OnInit called only once during lifecycle.

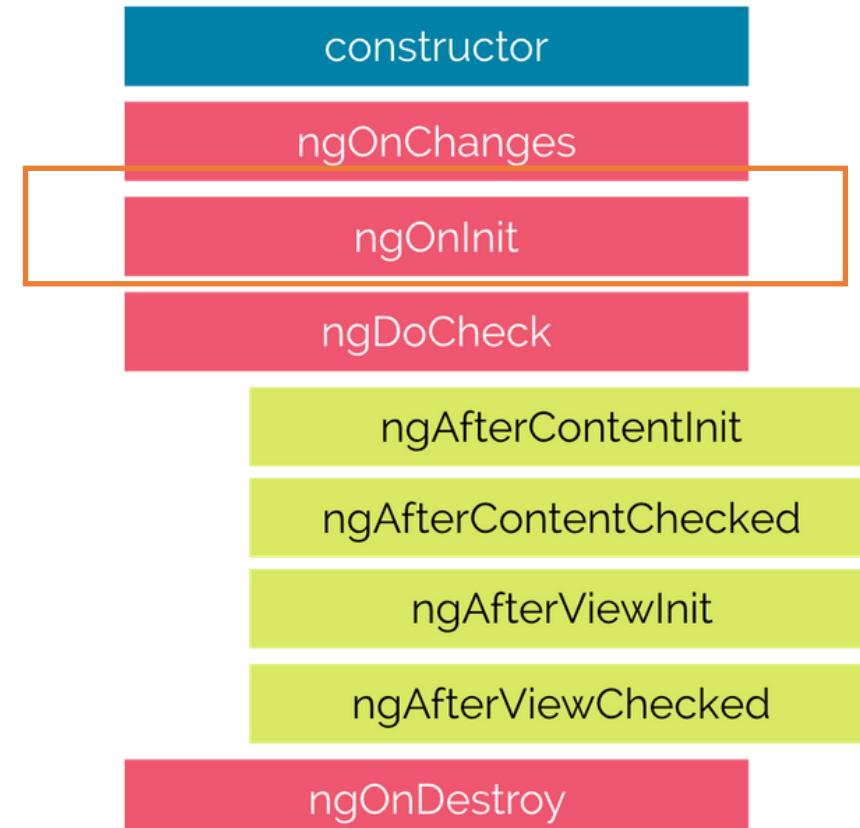
4. By default, present in the component.

```
@Component({
  selector: 'app-sample',
  templateUrl: './sample.component.html',
  styleUrls: ['./sample.component.css']
})
export class SampleComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```



ngOnInit	Constructor
1. ngOnInit is an Angular lifecycle hook , which signals the activation of the created component.	The constructor is a method in a TypeScript class , that automatically gets called when the class is being instantiated.
2. ngOnInit is called after ngOnChanges lifecycle-hook.	Constructor is called before any lifecycle-hook.
3. When ngOnInit called, everything about component is already ready, so it's use is to perform most of the business logic on component.	When constructor called, everything in component is not ready, so it's mostly used for injecting dependencies only.



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 8: Routing

Q45. What is **Routing**? How to setup **Routing**?

Q46. What is **router outlet**?

Q47. What are **router links**?

- ❖ Routing helps in navigating from one view to another view with the help of URL.

See here we have two Component links, Component 1 and Component 2 in App-Component.

Now when I click Component 1 link, then the url is updated to component1 and the component displayed is component 1.

Then when I click Component 2 link, then the url is updated to component2 and the component displayed is component 2.

So, in a single page we are navigating from one component to another component, that is routing.



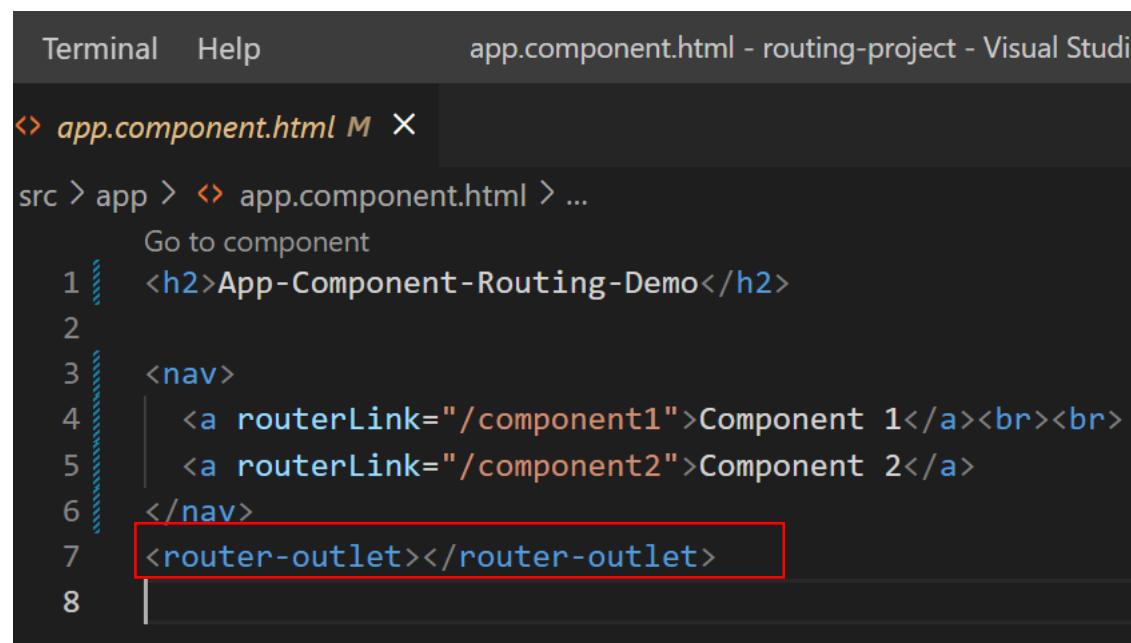
App-Component-Routing-Demo

Component 1

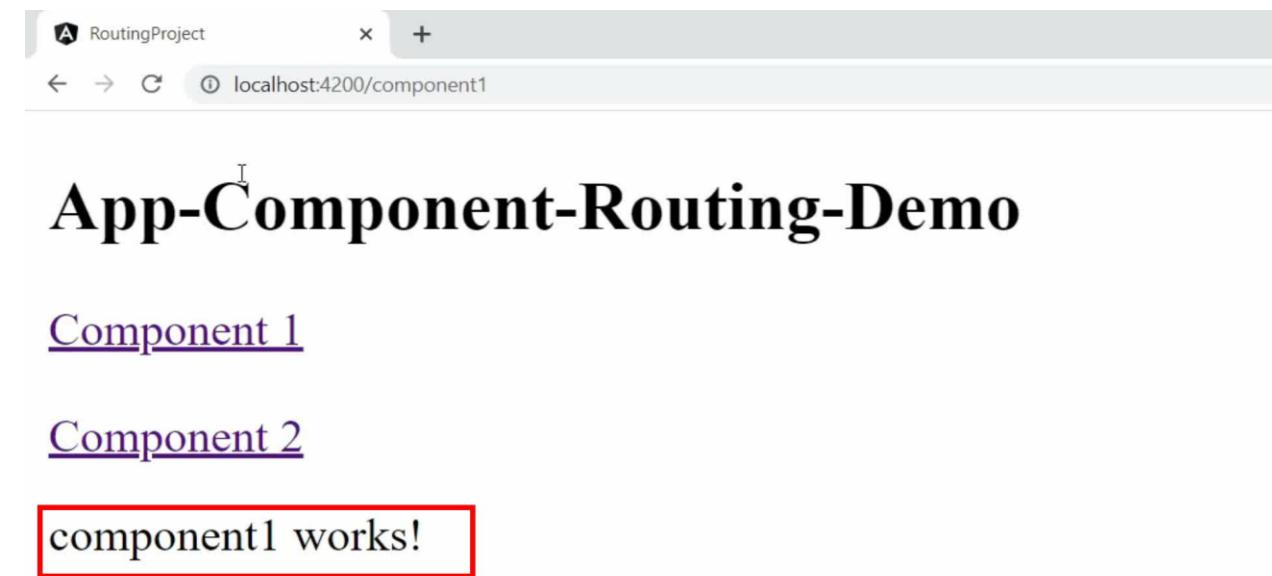
Component 2

component1 works!

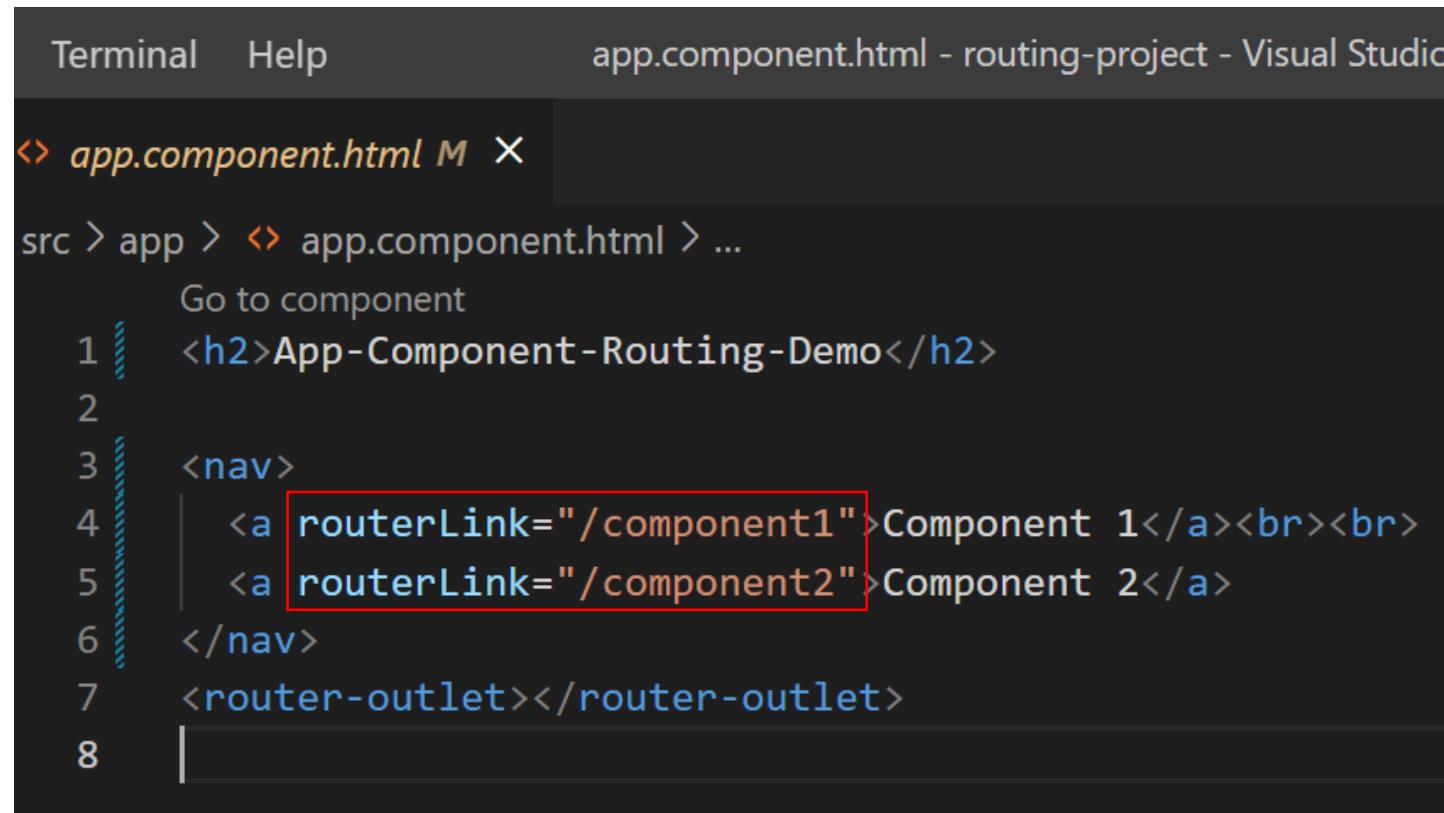
- ❖ Router-outlet in Angular works as a **placeholder**, which is used to load the different components dynamically based on the activated component via Routing.



```
Terminal  Help      app.component.html - routing-project - Visual Studio Code
<> app.component.html M X
src > app > <> app.component.html > ...
  Go to component
1   <h2>App-Component-Routing-Demo</h2>
2
3   <nav>
4     <a routerLink="/component1">Component 1</a><br><br>
5     <a routerLink="/component2">Component 2</a>
6   </nav>
7   <router-outlet></router-outlet>
8
```



- ❖ RouterLink is used for navigating to a different route.



```
Terminal  Help      app.component.html - routing-project - Visual Studio Code
<> app.component.html M X

src > app > <> app.component.html > ...
      Go to component
1  <h2>App-Component-Routing-Demo</h2>
2
3  <nav>
4  |  <a routerLink="/component1">Component 1</a><br><br>
5  |  <a routerLink="/component2">Component 2</a>
6  </nav>
7  <router-outlet></router-outlet>
8
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 9: Observable\ HttpClient\ RxJS

Q48. What are **Asynchronous** operations?

Q49. What is the difference between **Promise** and **Observable**?

Q50. What is **RxJS**?

Q51. What is **Observable**? How to implement **Observable**

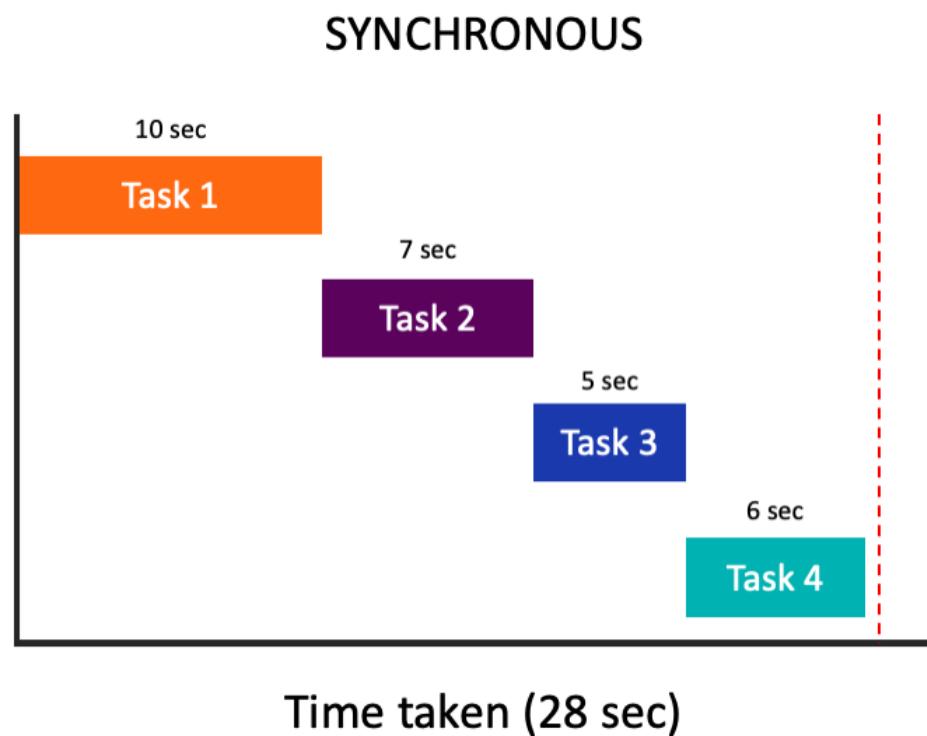
Q52. What is the role of **HttpClient** in Angular?

Q53. What are the steps for fetching the data with **HttpClient & Observable**?

Q54. How to do **HTTP Error Handling** in Angular?

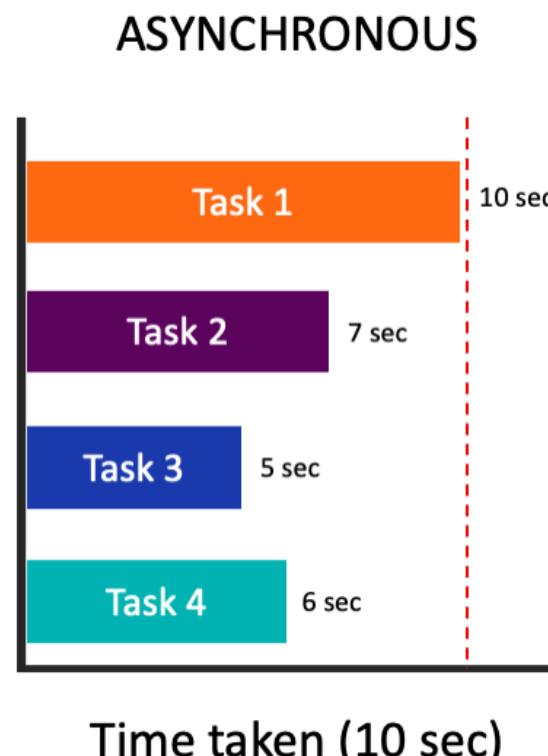
- ❖ In synchronous operations, tasks are executed in sequence and one by one.

*In image, you can see Task1, Task2, Task3, Task4 are executing **one by one** and taking in total more time 28 sec*



- ❖ In Asynchronous operations, tasks are executed in sequence and one by one.

*In image, you can see Task1, Task2, Task3, Task4 are executing **parallelly** and taking in total less time 10 sec*



- ❖ How to do asynchronous operations in Angular?

Observables are used to perform asynchronous operations and handle asynchronous data

Observables	Promises
1. Emit multiple values over a period of time. Also called streaming of data.	Emit a single value at a time.
2. Are lazy: they're not executed until we subscribe to them using the <code>subscribe()</code> method.	Are not lazy: execute immediately after creation.
3. Have subscriptions that are cancellable using the <code>unsubscribe()</code> method.	Are not cancellable .

❖ What is the advantage of Observable over Promises?

1. In Observable, streaming of data will display continuous data to end user. So, the user has not to wait.

In Promises, end user have to wait until the whole data is not available, which is not good.

2. Observables are lazy, means they will not execute until someone subscribe them, which is a good thing.

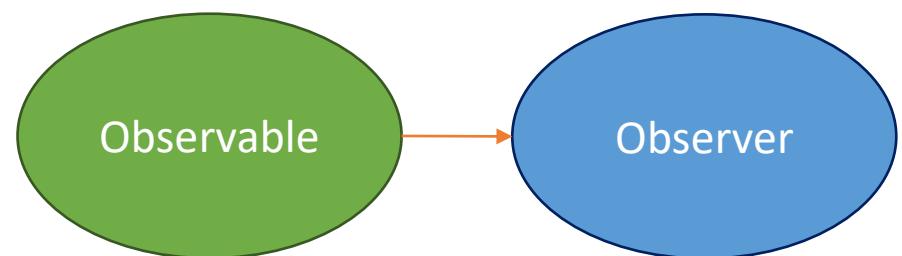
Promises are not lazy, means they will immediately execute even if there is no subscriber, which is not good.



The screenshot shows the RxJS website's 'Introduction' page. The header features the RxJS logo and navigation links for 'OVERVIEW', 'REFERENCE', and 'TEAM'. The main content area is titled 'Introduction' and contains a paragraph describing RxJS as a library for composing asynchronous and event-based programs using observable sequences. The RxJS logo is highlighted with a red border.

RxJS is a library for composing asynchronous and event-based programs by using observable sequences. It provides one core type, the `Observable`, satellite types (Observer, Schedulers, Subjects) and operators inspired by `Array` methods (`map`, `filter`, `reduce`, `every`, etc) to allow handling asynchronous events as collections.

- ❖ In simple words - RxJS is a javascript library, that allow us to work with asynchronous data stream with the help of observables.
- ❖ Observables are introduced by RxJS library.
- ❖ RxJS stands for Reactive Extensions for JavaScript.

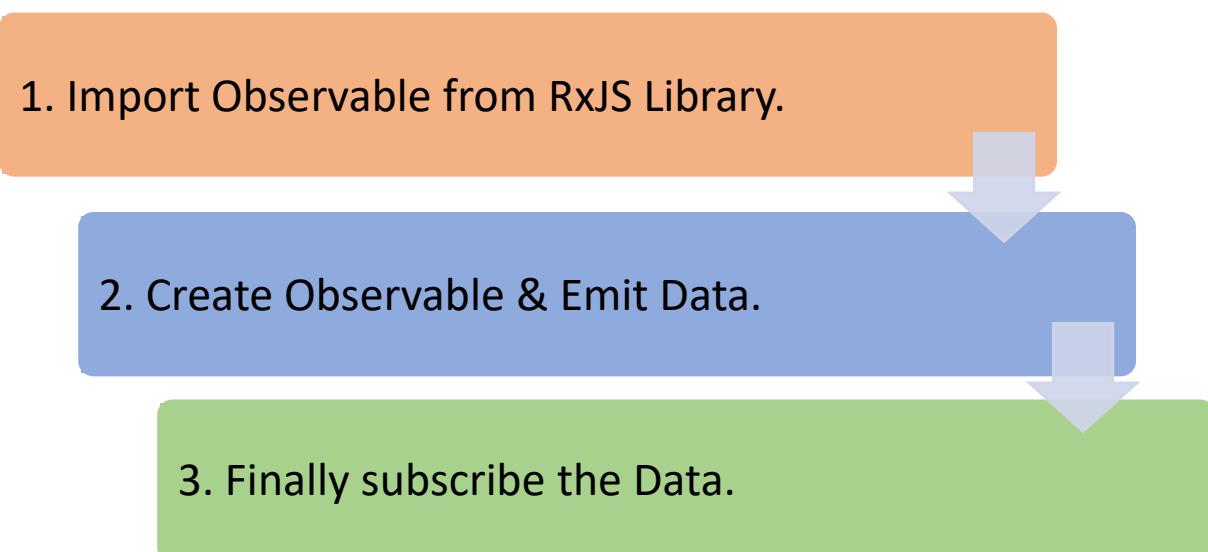


Stream of Data

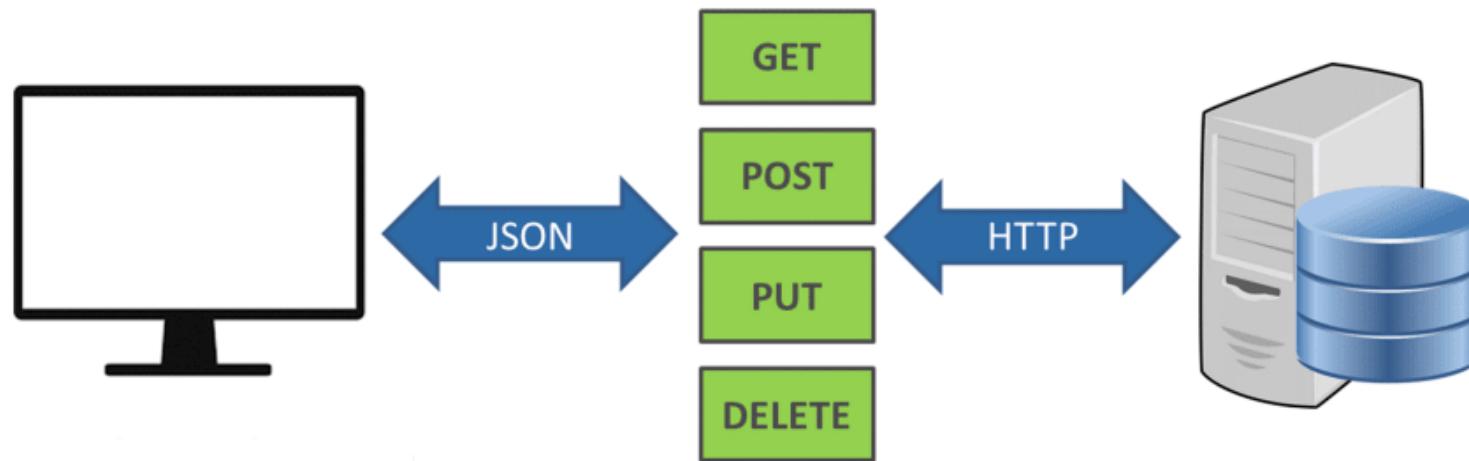
Subscriber

[Back to chapter index](#)

- ❖ Observables are used to **stream data** to multiple components.
- ❖ Basically, to receive the data from API's and continuously stream it to the multiple components.
- ❖ It's a 3 step process:



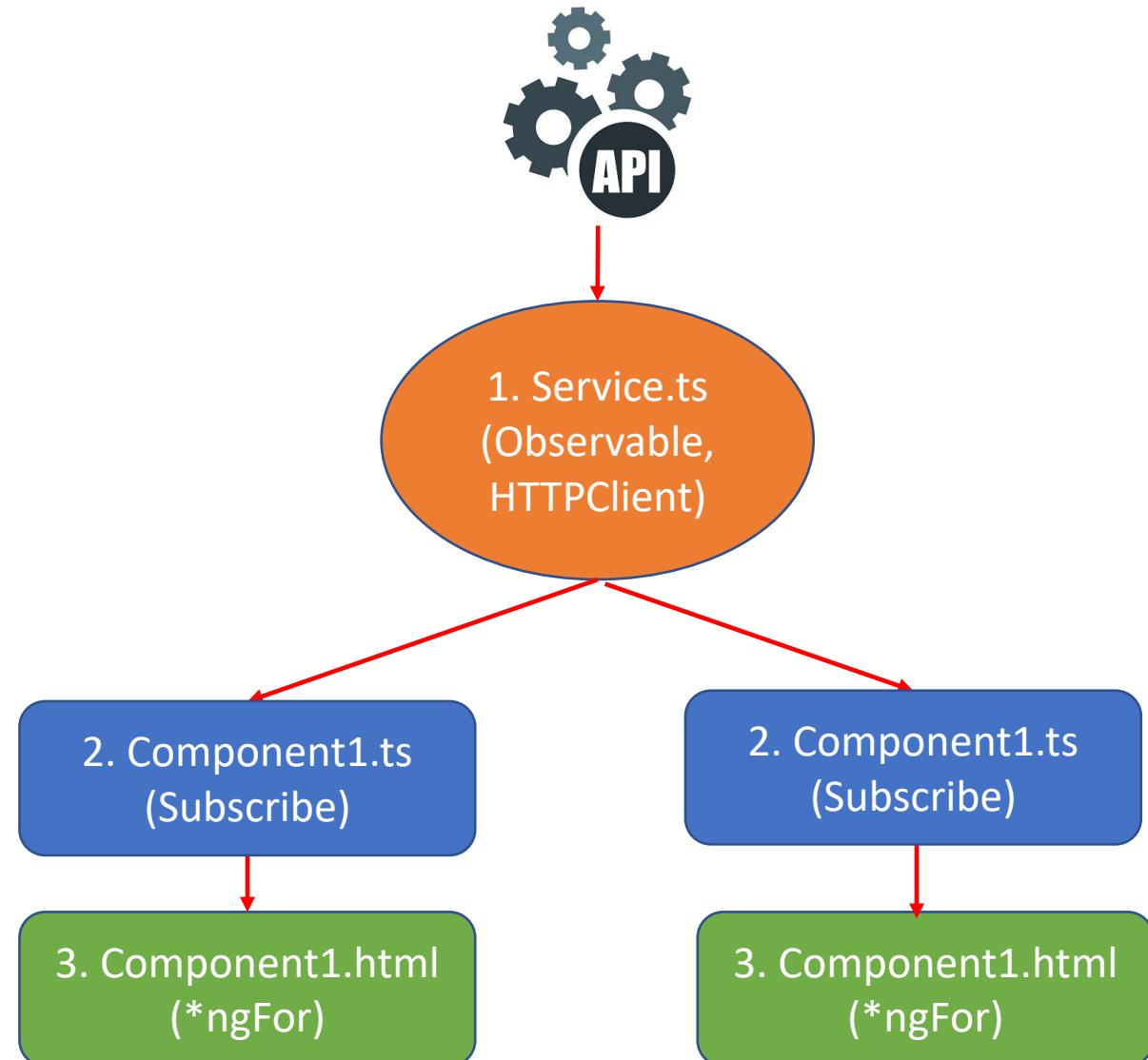
- ❖ HttpClient is a **built-in service** class provided by Angular, which is used to perform HTTP requests(GET/POST/PUT/DELETE) to send and receive data from API or Servers.
- ❖ Package name is - @angular/common/http package



- ❖ 3 Steps to fetch data with HttpClient & Observables:

Suppose we want to fetch a list of students from API.

1. Create a service and configure observable and HttpClient to receive the data from the API.
2. In the component class file, use Subscribe method to receive the data sent by the observable.
3. Finally in component HTML file, use the for loop to iterate and display the data list.



1. Create a service and configure observable and HttpClient to receive the data from the API.

```
@Injectable({
  providedIn: 'root'
})
export class StudentService {

  private _url: string = "assets/data/students.json";

  constructor(private http: HttpClient) { }

  getStudents(): Observable<IStudent[]>{
    return this.http.get<IStudent[]>(this._url);
  }
}
```

2. In the component class file, use Subscribe method to receive the data sent by the observable.

```
export class AppComponent implements OnInit{  
  public students : any[] = [];  
  
  constructor(private _studentService: StudentService) {  
  }  
  
  ngOnInit()  
  {  
    this._studentService.getStudents()  
    .subscribe(data => this.students = data);  
  }  
}
```

2. Finally in component HTML file, use the for loop to iterate and display the data list.

```
<ul *ngFor="let student of students">
  <li>Student Id - {{student.id}}</li>
  <li>Student Name - {{student.name}}</li>
</ul>
```

- ❖ HTTP Error Handling can be done by using **catchError** and **throwError** functions from rxjs.

```
//The pipe is used for chaining the fucntions,  
//which will execute in sequence then.  
return this.http.get<IStudent[]>(this._url).pipe(  
  catchError((error) => {  
    return throwError(() => error);  
  })  
)
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 10: Typescript-Basics

Q55. What is Typescript? Or What is the difference between Typescript and Javascript?

Q56. How to install Typescript and check version?

Q57. What is the difference between let and var keyword?

Q58. What is Type annotation?

Q59. What are Built in/ Primitive and User-Defined/ Non-primitive Types in Typescript?

Q60. What is “any” type in Typescript?

Q61. What is Enum type in Typescript?

Q62. What is the difference between void and never types in Typescript?

Q63. What is Type Assertion in Typescript?

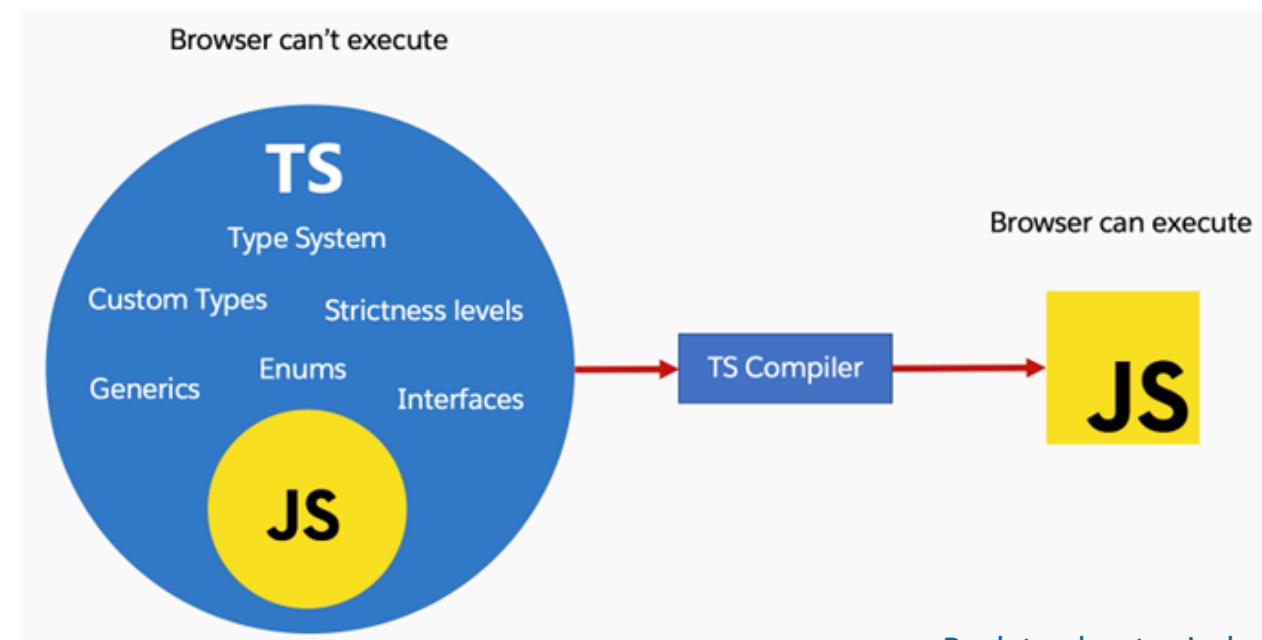
Q64. What are Arrow Functions in Typescript?

[Back to main index](#)

- ❖ Typescript is an open-source programming language.
It's advantage are:

1. Typescript is a strongly typed language.
2. Typescript is a superset of JavaScript.
3. It has Object oriented features.
4. Detect error at compile time.

- ❖ In image, you can see TS(Typescript) contains JS(Javascript) and other additional features(type system, custom types, generics, enums, interfaces).
- ❖ Browser can't execute typescript, so finally TS Compiler will convert the TS to JS only, which browser can understand.



- ❖ Here is the command which can be used to install typescript on your system (without Angular).

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

C:\Users\████████td\

C:\>npm install -g typescript
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

changed 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
```

- ❖ var and let are both used for variable declaration, but the difference between them is that **var is global function scoped and let is block scoped**.

Compile time error, since let variable “i” scope is only limited inside for loop.

No error, since var variable “j” scope is not limited inside for loop, but inside the whole function fnLetVar().

```
function fnLetVar()
{
    //let keyword
    for(let i=0; i<5; i++)
    {
        console.log("Inside " + i);
    }
    console.log("Outside " + i);

    //var keyword
    for(var j=0; j<5; j++)
    {
        console.log("Inside " + j);
    }
    console.log("Outside " + j);
}

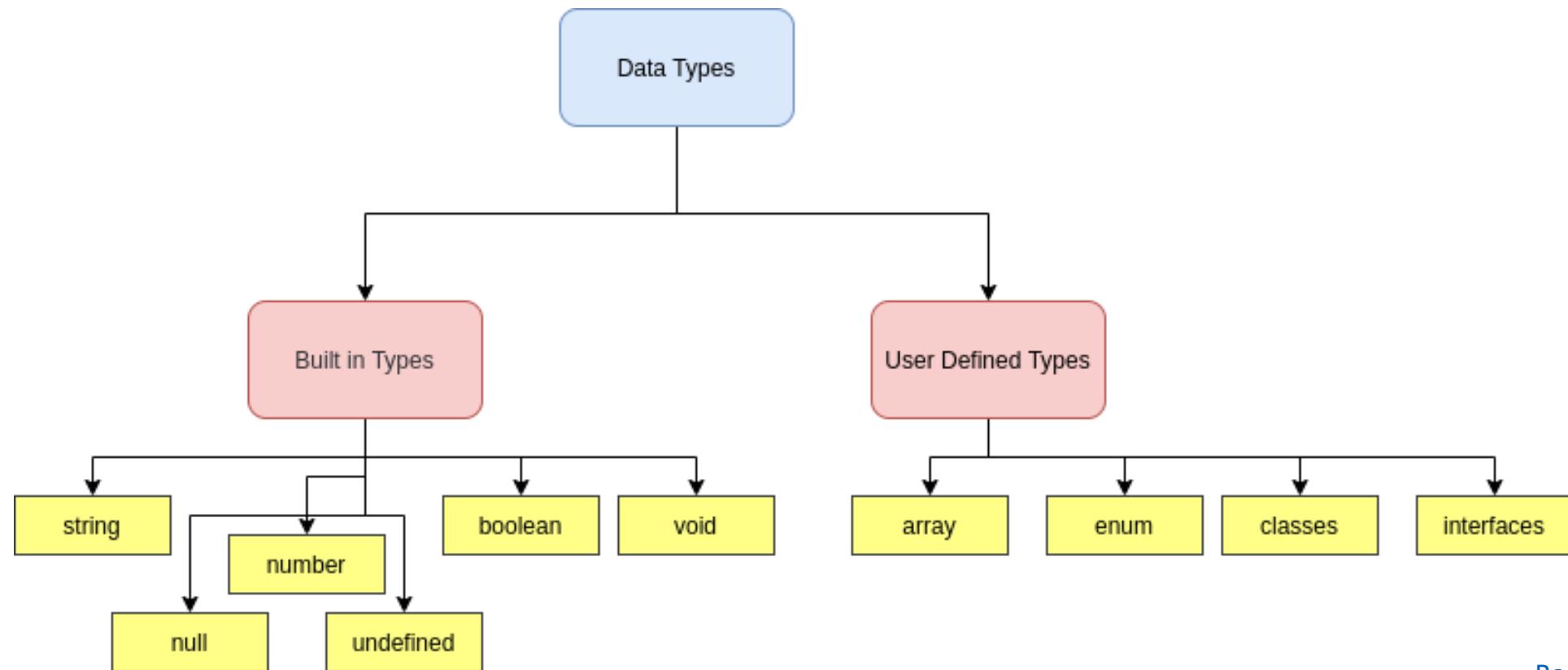
fnLetVar();
```

- ❖ Type annotations helps the compiler in checking the types of variable and avoid errors.

For example, in below image when we set variable “i” as number. And now if you assign “i” a string or Boolean value, then Typescript will show the compile time error, which is good.

```
let i : number;  
  
i = 1;  
  
i = "Happy";  
  
i = true;
```

- ❖ Built-in Data types are used to store simple data. Such as string, number, Boolean.
- ❖ User-Defined types are used to store complex data. Such as array, enum, classes.



- ❖ When a value is of type any, then **no typechecking(no compile time error)** will be done by compiler and the flexibility is there to do anything with this variable.

For example, see in the image there is no compile time error for assigning different types to variable “x” because it is of any type.

```
let x: any;

x = 1;

x = "Happy";

x = true;

x();

x.AnyProperty = 10;
```

- ❖ Enums allows to define a set of **named constants**.

Suppose you have a family of constants(Directions) like this in image.

Now if you want to insert some direction, for example SouthWest between 1 and 2, then it will be a problem, as you have to change this in all of your application.

So here enum is a better way as you don't have to assign the numbers then.

```
const DirectionNorth = 0;
const DirectionSouth = 1;
const DirectionWest = 2;
const DirectionEast = 3;

let dir = DirectionEast;
```

```
enum Direction {
  North,
  South,
  West,
  East
}

let dirNew = Direction.East;
```

- ❖ void means no type. It is used when the function will return empty.

In image, fnVoid() function will return empty.

```
let message = "Happy";

function fnVoid(message: string): void {
    console.log("void " + message);
}
```

- ❖ never means it will return never. It is used to throw error only.

For example, in image fnNever function which will not reach to its last line, and it always throws an exception.

```
function fnNever(message: string): never {
    throw new Error(message);
}
```

- ❖ Type assertion is a technique that informs the compiler about the type of a variable.

For example, in below image, if we do not put `<String>`, then typescript will try to assume the type of “secondname” by itself automatically.

But by putting `<String>`, we are asserting and informing the compiler about the type of `secondName` as `String` and now compiler will not do automatic typechecking.

```
let myname: any = "Happy";  
  
// Conversion from any to string  
let secondName = <String> myname;  
  
console.log(secondName);  
console.log(typeof secondName);
```

- ❖ An **arrow function** expression is a compact alternative to a traditional function expression.

In below image, you can see how the same function can be written using arrow function.

```
//Normal function approach
let x = function(a, b)
{
    a * b;
}

//Arrow function approach
let y = (a, b) => a * b;
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 11: Typescript - OOPS

Q65. What is Object Oriented Programming in Typescript?

Q66. What are Classes and Objects in Typescript?

Q67. What is Constructor?

Q68. What are Access Modifiers in Typescript?

Q69. What is Encapsulation in Typescript?

Q70. What is Inheritance in Typescript?

Q71. What is Polymorphism in Typescript?

Q72. What is Interface in Typescript?

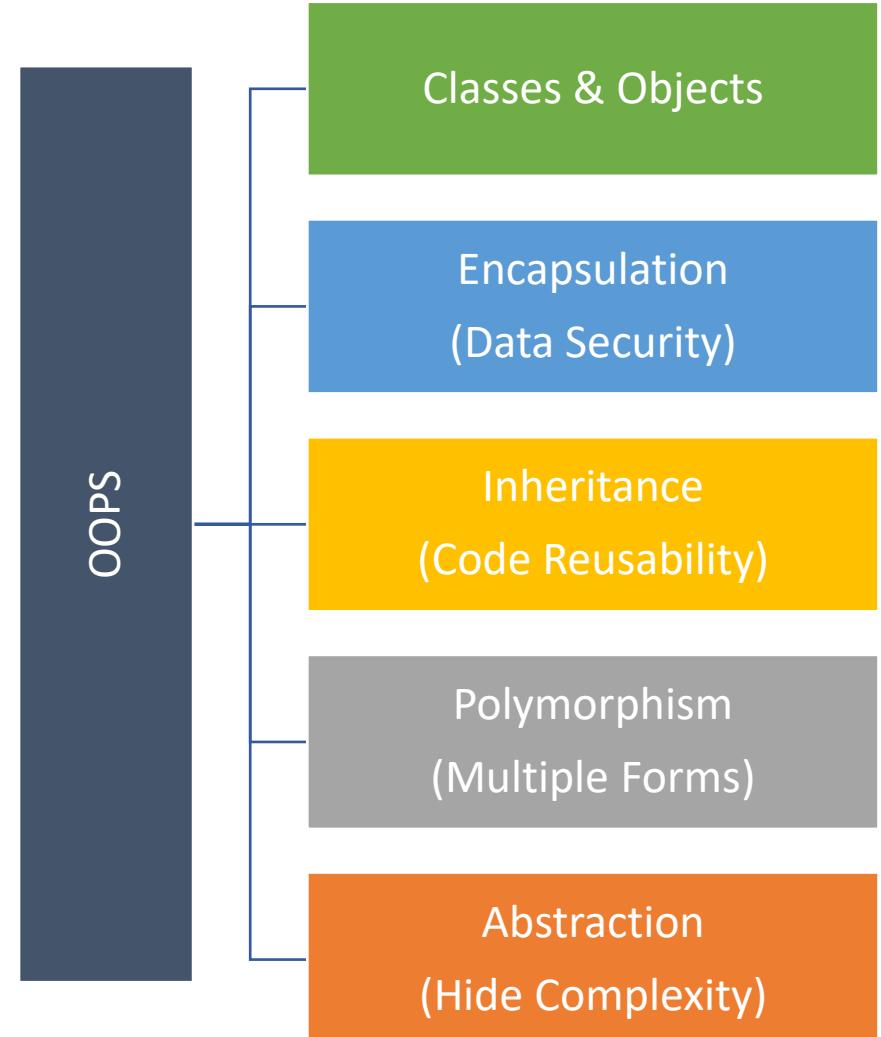
Q73. What's the difference between extends and implements in TypeScript

Q74. Is Multiple Inheritance possible in Typescript?

[Back to main index](#)

- ❖ Object oriented programming is used to design structured and better applications.

In the image, you can see the concepts used in object oriented programming.



- ❖ Classes are blueprint for individual objects.
- ❖ Objects are instances of a class.

Person is a class with two properties(name, age) and one method(familyCount)

objPerson is the object/instance of Person class. By this object only we can set Person class properties and call the method of the class.

```
class Person {  
  name: string;  
  age: number;  
  
  familyCount(){  
    return 6;  
  }  
}
```

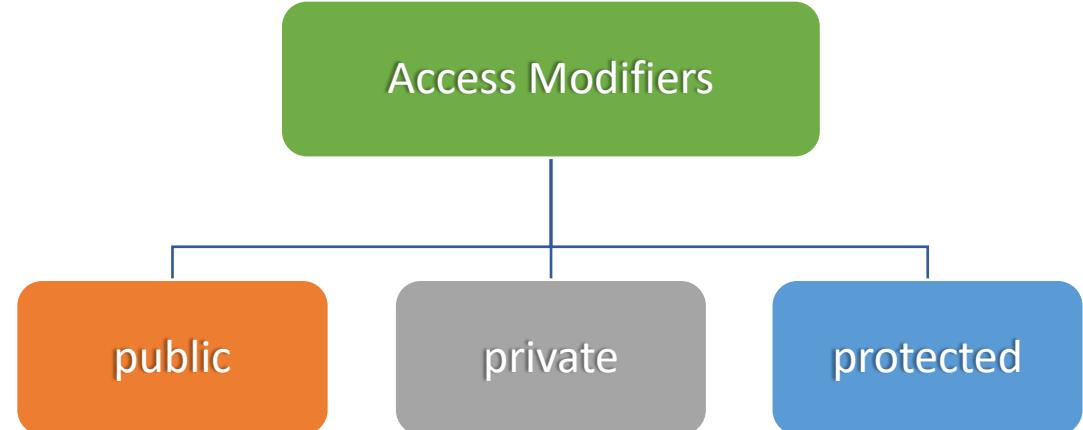
```
var objPerson = new Person();  
  
let myName = objPerson.name = "Happy";  
let myAge = objPerson.age = 10;  
  
let countFamily = objPerson.familyCount();
```

- ❖ The constructor is a method in a **TypeScript class** that automatically gets called when the class is being **instantiated**.
- ❖ Constructor always run before any angular hook and it is not a part of Lifecycle Hooks.
- ❖ Constructor is widely used to inject **dependencies(services)** into the component class.

```
ts sample.component.ts U ×
src > app > sample > ts sample.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-sample',
5    templateUrl: './sample.component.html',
6    styleUrls: ['./sample.component.css']
7  })
8  export class SampleComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14
15 }
16
```

❖ 3 types of Access modifiers in Typescript:

1. The public modifier allows class variable, properties and methods to be accessible from all locations.
2. The private modifier limits the visibility to the same-class only.
3. The protected modifier allows properties and methods of a class to be accessible within same class and within subclasses/derived/child.



1. empId no error, as it is public.

2. empName error, as it is out of the class and it is private.

3. empAge error, as it is out of the class and it is protected.

3. But in PermanentEmployee empAge no error, because it is inside the derived class(PermanentEmployee) of Employee.

```
class Employee
{
    public empId: number;
    private empName: string;
    protected empAge: number;
}

var objEmployee = new Employee();
objEmployee.empId = 10;
objEmployee.empName = "Amit";
objEmployee.empAge = 100;

class PermanentEmployee extends Employee
{
    empId = 100;
    empAge = 30;
}
```

- ❖ Encapsulation is the wrapping up of data and function together to access the data.

For example, here in code `_empId` is the data. We can make it public and then it can be accessible outside the Employee class.

But as per OOPS concept encapsulation this is wrong as it can lead to data security issues.

Therefore, we should mark `this._empId` data as private. And then create a function(`getEmpId`), which will help in accessing `this._empId` outside of this class.

Outside the class, in the last line we are calling `getEmpId` method to access/display the data `_empId`.

- ❖ What is the advantage of Encapsulation?

This is good for data security to prevent direct data access.

```
class Employee
{
    private _empId: number; //data

    getEmpId(){
        return this._empId;
    }
}

let objEmployee = new Employee();

console.log(objEmployee.getEmpId());
```

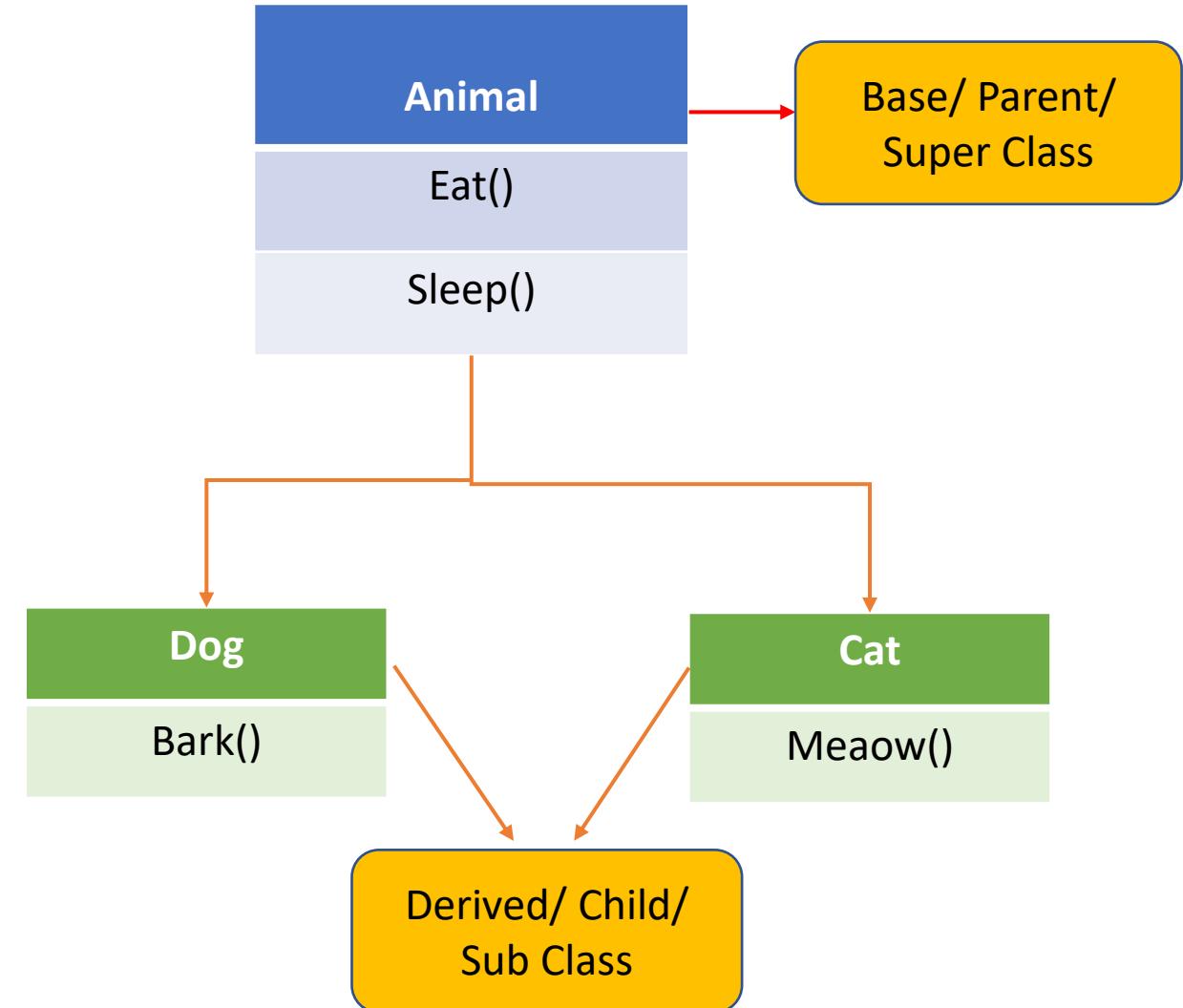
- Inheritance is a feature where derived class will acquire the properties and methods of base class.

For example, in the image Dog and Cat class are derived from Animal base class.

Therefore, Eat() and Sleep() methods of Animal base class will be automatically available inside Dog and Cat derived class without even writing it.

- What is the advantage of Inheritance?

Inheritance is good for reusability of code. As you can write one method in base class, and then use it in multiple derived classes.



```
class Animal{  
    Eat(){  
        console.log("eat");  
    }  
}  
  
class Dog extends Animal{  
    Bark(){  
        console.log("bark");  
    }  
}  
  
let objectDog = new Dog();  
  
objectDog.Eat();  
  
objectDog.Bark();
```

Both Eat() and Bark() method are accessible via Dog class object. Eat is from the base class but.

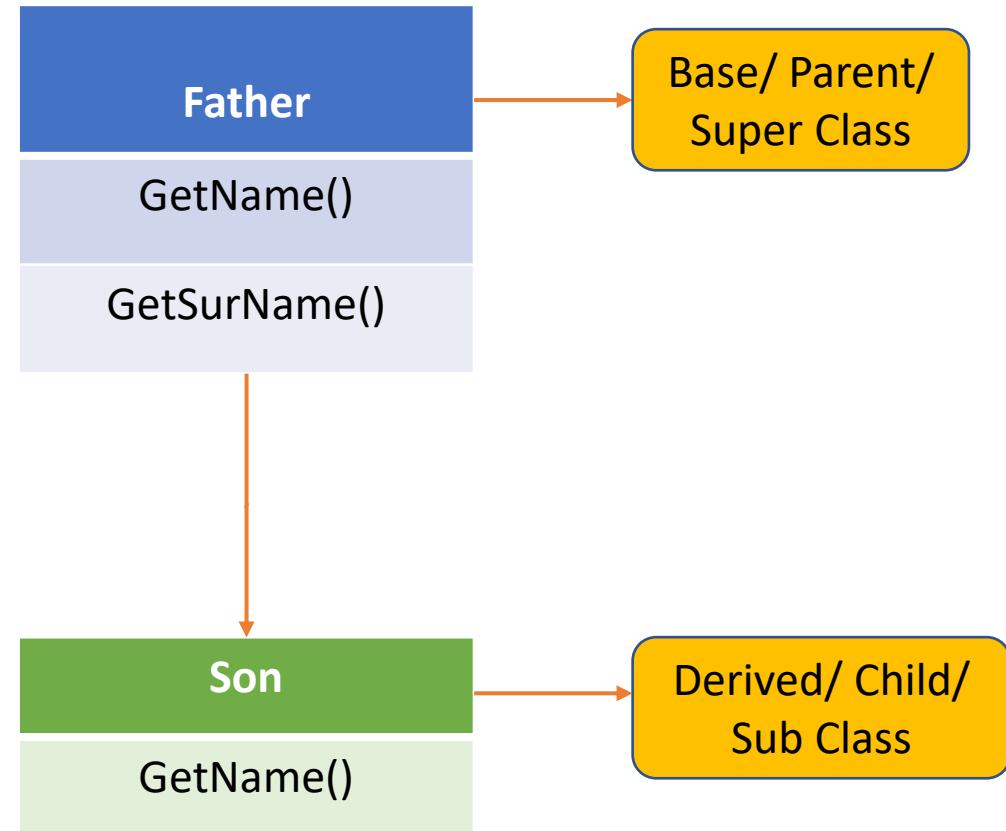
- ❖ Polymorphism means same name method can take multiple Forms.

For example, in the image Son can inherit the GetSurName() method implementation of father. But the GetName() method implementation should be different for son. So son can have a different implementation but the method name can be same as GetName().

- ❖ What is the advantage of Polymorphism?

Like in dictionary, a word “Running” can have multiple meanings: Running a race or Running a computer.

Similarly in programming, Polymorphism can be used to do multiple things with the same name.



GetName() method of Father class object output is different, and Getname() method of Son class object output is different. Same name method taking multiple forms.

```
class Father {  
    GetName() {  
        console.log('Amit');  
    }  
    GetSurName() {  
        console.log('Singh');  
    }  
}  
  
class Son extends Father {  
    GetName() {  
        console.log('Ravi');  
    }  
}  
  
let objectFather = new Father();  
objectFather.GetName();  
//Output: Amit  
  
let objectSon = new Son();  
objectSon.GetName();  
//Output: Ravi
```

- ❖ An interface is a contract where methods are only declared.
- ❖ What is the advantage of creating Interface?

For example, we have multiple classes *PermanentEmployee*, *ContractEmployee* etc.

They all must have the *Role()* method, but the body of *Role()* method can be different.

Now to maintain consistency we will create the Interface *IEmployee*, with just declaration of *Role()* method and implement this interface in all the classes.

This will make sure all the Employees classes are in sync with the *Role* method. It will also help in unit testing.

- ❖ One advantage is, multiple inheritance can be achieved via interfaces only.
- ❖ Also, if we create interfaces for the classes in application then unit testing is very easy, otherwise it is difficult.

```
interface IEmployee
{
    Role(): void;
}

class PermanentEmployee implements IEmployee{
    Role(){
        console.log("Lead");
    }
}

class ContractEmployee implements IEmployee{
    Role(){
        console.log("Developer");
    }
}
```

- ❖ **Extends** used for base class inheritance.

```
//Parent Class
class Employee{
    Salary()
    {
        console.log("salary");
    }
}

//Child Class
class PermanentEmployee extends Employee {
```

- ❖ **Implements** used for interface inheritance.

```
interface IEmployee{
    ...
}

class PermanentEmployee implements IEmployee {
    Salary() {
        ...
    }
}
```

- ❖ Multiple inheritance means when one derived class is inherited from multiple base classes.

- ❖ Multiple inheritance in typescript only possible via Interfaces.

See in first image there is the compile time error in Employee2 as multiple base classes are not allowed, but in second image with multiple interfaces it is working.

```
//Parent Class 1
class Employee1{
    Salary1()
    {
        console.log("salary");
    }
}

//Parent Class 2
class Employee2{
    Salary2()
    {
        console.log("salary");
    }
}

//Child Class
class PEmployee extends Employee1, Employee2 {
}
```

```
interface IEmployee1{
    Salary1();
}

interface IEmployee2{
    Salary2();
}

class TEmployee implements IEmployee1, IEmployee2 {
    Salary1() {
        console.log("300000");
    }
    Salary2() {
        console.log("500000");
    }
}
```



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 12: Angular Forms

Q75. What are Angular Forms? What are the type of Angular Forms?

Q76. What is the difference between Template Driven Forms & Reactive Forms?

Q77. How to setup Template Driven Forms?

Q78. How to apply Required field validation in template driven forms?

Q79. What is Form Group and Form Control in Angular?

Q80. How to setup Reactive Forms?

Q81. How to do validations in reactive forms?

- ❖ **Angular forms** are used to handle user's input.

For example, login form, registration form, feedback form, update profile form etc etc.

All these are forms, in which user will input and then submit the information.

Log in

Sorry, That login is incorrect

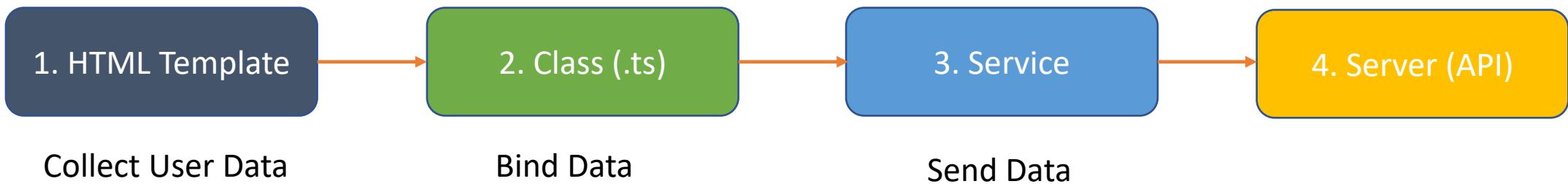
Username:

Password:

Log In

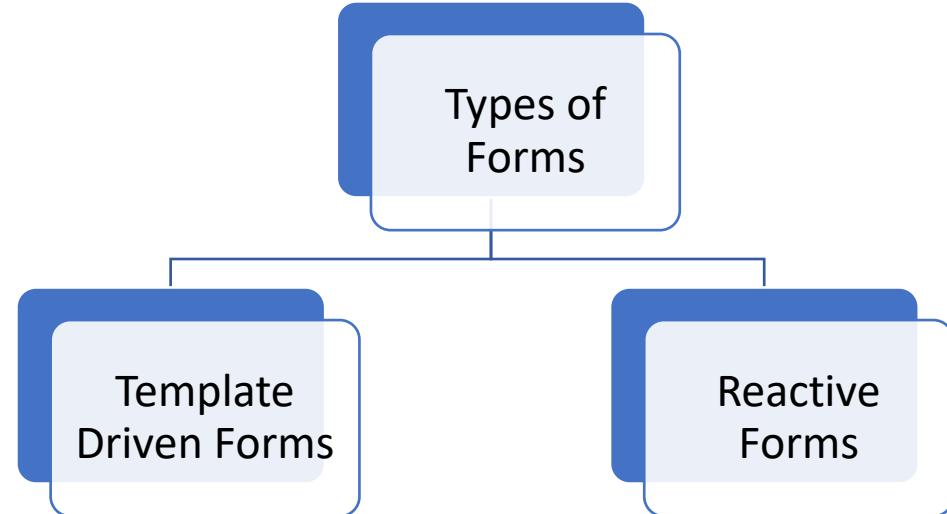
❖ 4 steps data flow process in Angular forms-

1. The user will input the data in the HTML template file. .
2. Then data will be bind in the component typescript class file.
3. Then it will be sent to the Angular service.
4. Angular service will then finally send it to the Server which will receive the data and process the data. (In most cases this server will be an API only.)



❖ There are 2 types of forms in Angular:

1. Template driven forms - In template driven forms, most of the code will be written and handled in HTML templates only.
2. Reactive forms - in reactive forms most of the code will be written in component typescript class file.



Template Driven Forms	Reactive Forms
1. Most code and validation logic is written in HTML template .	Most code and validation logic is written in component typescript class file.
2. Have to add FormsModule in AppModule to activate it.	Have to add ReactiveFormsModule in AppModule to activate it.
3. Used when application is simple and have less controls .	Used when application is complex and have more controls .

- ❖ Below is the sample code to setup login form using template driven form approach in html template file. Here we have used template reference variable (#loginForm).

```
<h3>Template Driven Form</h3>

<form #loginForm="ngForm" (ngSubmit)="login(loginForm.value)">

  <input type="text" required name="email" placeholder="Email" ngModel>
  <br><br>

  <input type="password" name="password" placeholder="Password" ngModel>
  <br><br>

  <button [disabled]="loginForm.invalid">Login</button>

</form>
```

- ❖ For example, if you want to put required field validation in email field, then put the required keyword there, which will make the form invalid until the email is empty. Also, you can disable the button if the form is invalid.

```
<h3>Template Driven Form</h3>

<form #loginForm="ngForm" (ngSubmit)="login(loginForm.value)">

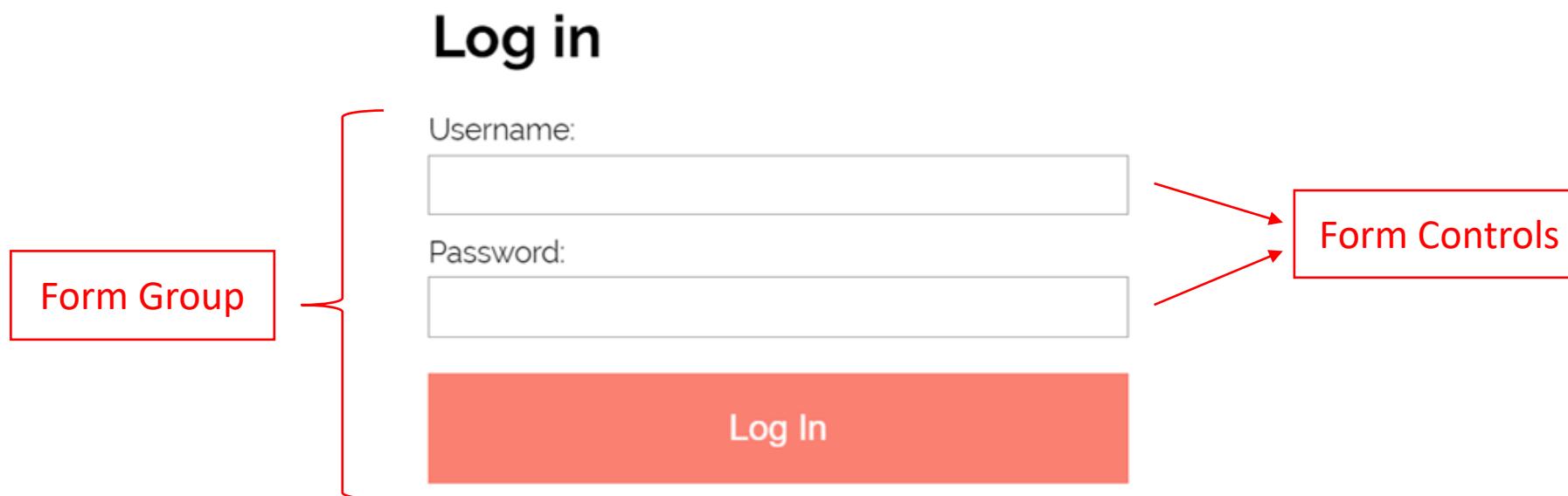
  <input type="text" required name="email" placeholder="Email" ngModel>
  <br><br>

  <input type="password" name="password" placeholder="Password" ngModel>
  <br><br>

  <button [disabled]="loginForm.invalid">Login</button>

</form>
```

- ❖ Form controls are the **individual** elements.
- ❖ Form groups is a **collection** of form controls.



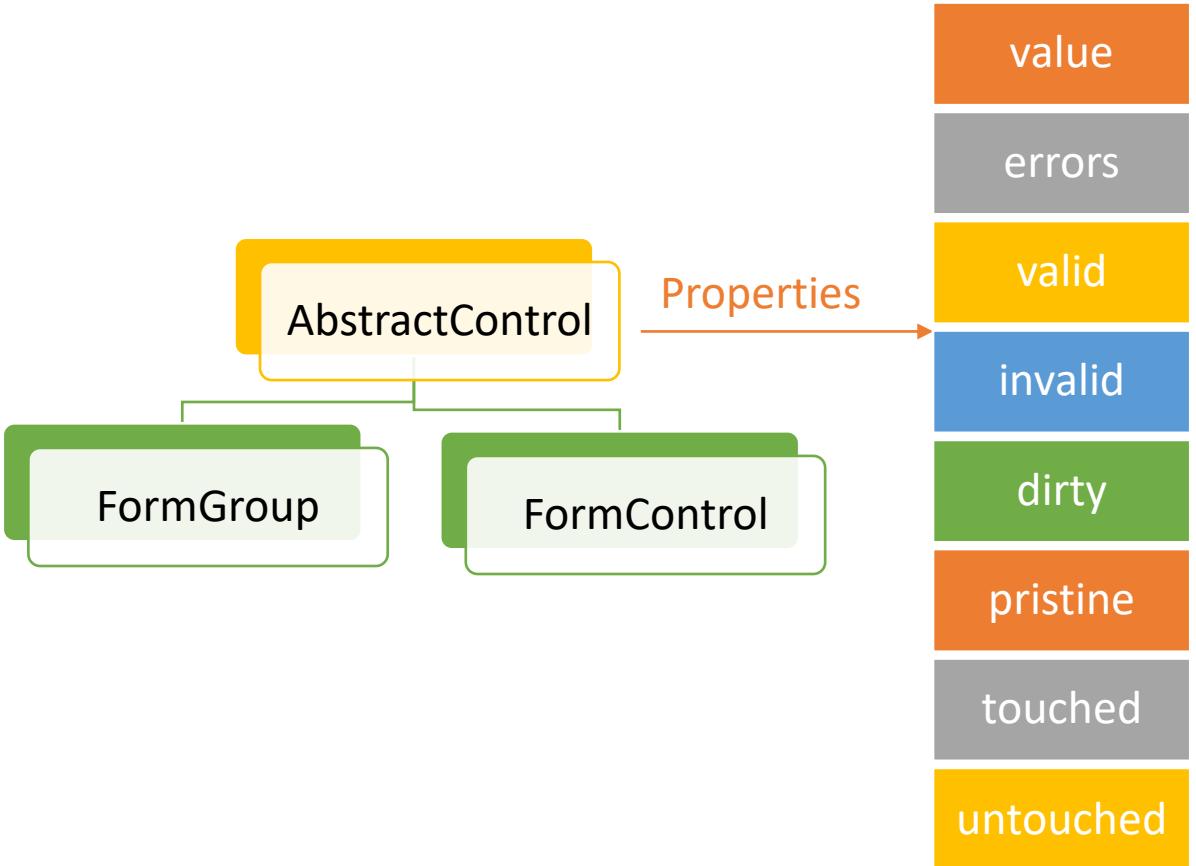
❖ What is the use of FormGroup and FormControl?

Both FormGroup and FormControl are used to **track** the values and states of controls.

In image you can see, both formgroup and formcontrol are inherited from AbstractControl class. AbstractControl class has these properties - Value, errors, valid, invalid, dirty, pristine, touched and untouched.

Now these properties will help in **tracking the changes in each and every control**. For example, value property will be the value of any input field.

That's why FormGroup and FormControl are required to keep track of the changes in the controls.



❖ Below are some important points about Reactive Forms:

1. In reactive forms most code and validation logic is written in **component typescript** file.
2. Reactive Forms can be activated by adding **ReactiveFormsModule** in AppModule.

❖ When to use Reactive forms and when to use template driven forms?

Reactive Forms are used when application is **complex** and have more number of controls.

Template driven forms is used when number of controls are very less in the form.

Normally we prefer reactive forms in general.

Log in

Username:

Password:

Log In

```
export class AppComponent{  
  
  loginForm = new FormGroup({  
    username: new FormControl(null, Validators.required),  
    password: new FormControl()  
  });  
  
  submitLogin(): void{  
    console.log(this.loginForm);  
  }  
  
  validateUsername()  
{  
  return this.loginForm.get('username');  
}  
}
```

Log in

Username:

Password:

Log In

```
export class AppComponent{  
  
  loginForm = new FormGroup({  
    username: new FormControl(null, Validators.required),  
    password: new FormControl()  
  });  
  
  submitLogin(): void{  
    console.log(this.loginForm);  
  }  
  
  validateUsername()  
{  
  return this.loginForm.get('username');  
}  
}
```

Log in

Username:

Username Required

Password:

Log In

TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 13: Authentication/ JWT/ Auth Guard/ HTTP Interceptor

Q82. What is Authentication & Authorization in Angular?

Q83. What is JWT Token Authentication in Angular?

Q84. How to Mock or Fake an API for JWT Authentication?

Q85. How to implement the Authentication with JWT in Angular?

Q86. What is Auth Guard?

Q87. What is HTTP Interceptor?

Q88. How to Retry automatically if there is an error response from API?

Q89. What are the parts of JWT Token?

Q90. What is Postman?

Q91. Which part of the request has the token stored when sending to API?

[Back to main index](#)

- ❖ Authentication is the process to verify the **user's identity**.
- ❖ Authorization is the process of allowing a user to **access** the resources as per the role.



- ❖ Authentication is always done before Authorization.

- ❖ JWT authentication is a process to authenticate Angular user credentials to get or post data from API's.

Upcoming is the image, where we will see, how the request will start from Angular and what are the steps to authenticate the request and get the data back from the API.



Front-end/ Client-side

4. Store JWT token
at local storage

1. POST: {username, password}



3. Return Response {JWT token}



5. Request Data {JWT token: Header}



7. Send Data



Middleware/ Server-side/ Backend

2. Authenticate &
create JWT Token6. Validate token
signature

Angular developer need not to develop the API. This question is better explained in the video.

❖ Steps to implement JWT Authentication in Angular:

1. Create a reactive login form, setup email and password.
2. Create one SubmitLogin method and check whether login form is valid or not.
3. If valid, call service(AuthService) method(proceedLogin) and send the loginForm values and use subscribe method to get the result from the observable.
4. Store the token in local storage of browser. So that for next subsequent request, only token is sent, not the credentials again.

```
  },
  export class LoginComponent {
    constructor(private service: AuthService, private route: Router) {
    }

    public loginForm = new FormGroup({
      email: new FormControl(),
      password: new FormControl()
    });

    SubmitLogin(): void{
      if(this.loginForm.valid)
      {
        this.service.proceedLogin(this.loginForm.value).subscribe(result => {
          if(result != null)
          {
            localStorage.setItem('token', result.access_token);
            this.route.navigate(['home']);
          }
        });
      }
    }
  }
}
```

Go to Line/Column

[Back to chapter index](#)

5. Create Service(AuthService).
6. Inside service set API URL(apiurl)
7. Setup HttpClient to send the request using API url.
8. Use Observable to receive the data or token from the request and return it to the component.

❖ Why we created service and have not directly written this code in component?

So that all the components can use this reusable service method.

```
export class AuthService {  
  apiurl = 'http://localhost:8000/auth/login';  
  
  constructor(private http: HttpClient) { }  
  
  proceedLogin(user: any): Observable<any> {  
    return this.http.post(this.apiurl, user);  
  }  
  
  isLoggedIn(): boolean {  
    return localStorage.getItem('token') != null;  
  }  
  
  getToken(): string {  
    return localStorage.getItem('token');  
  }  
}
```

- ❖ Here are the Angular features we have used to implement Authentication in Angular:

1. Services

2. Routing

3. Reactive Forms

4. HttpClientModule

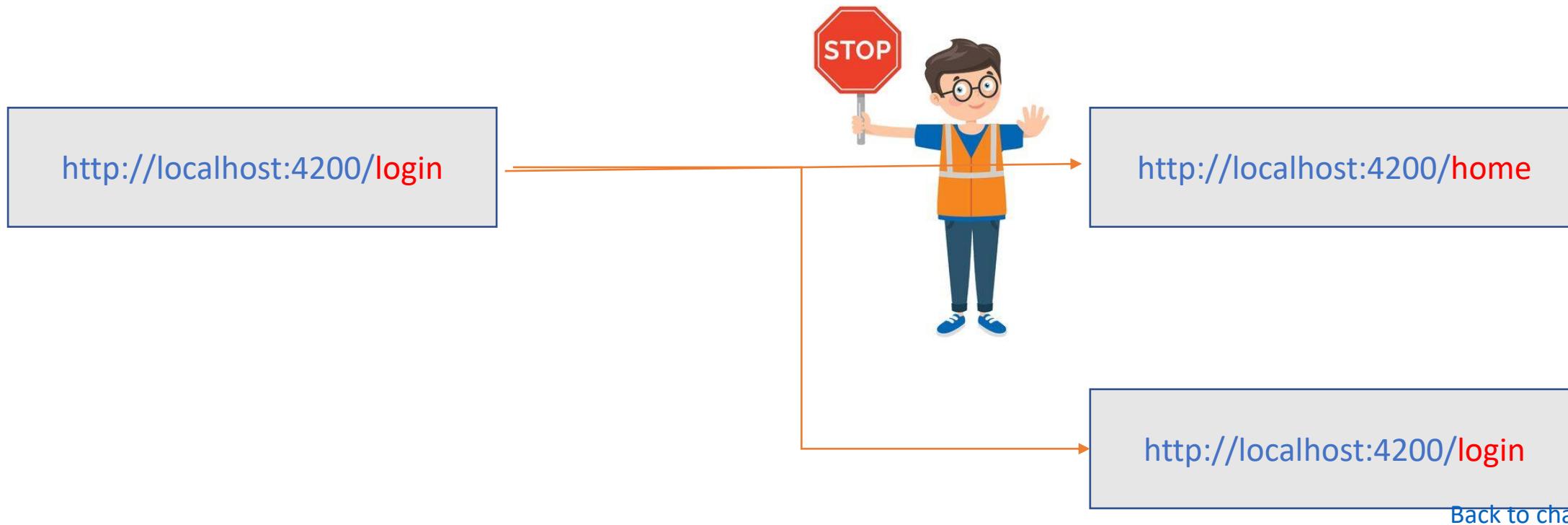
5. Dependency Injection

6. Observable & Subscriber

- ❖ Auth guard is used for guarding the routes.

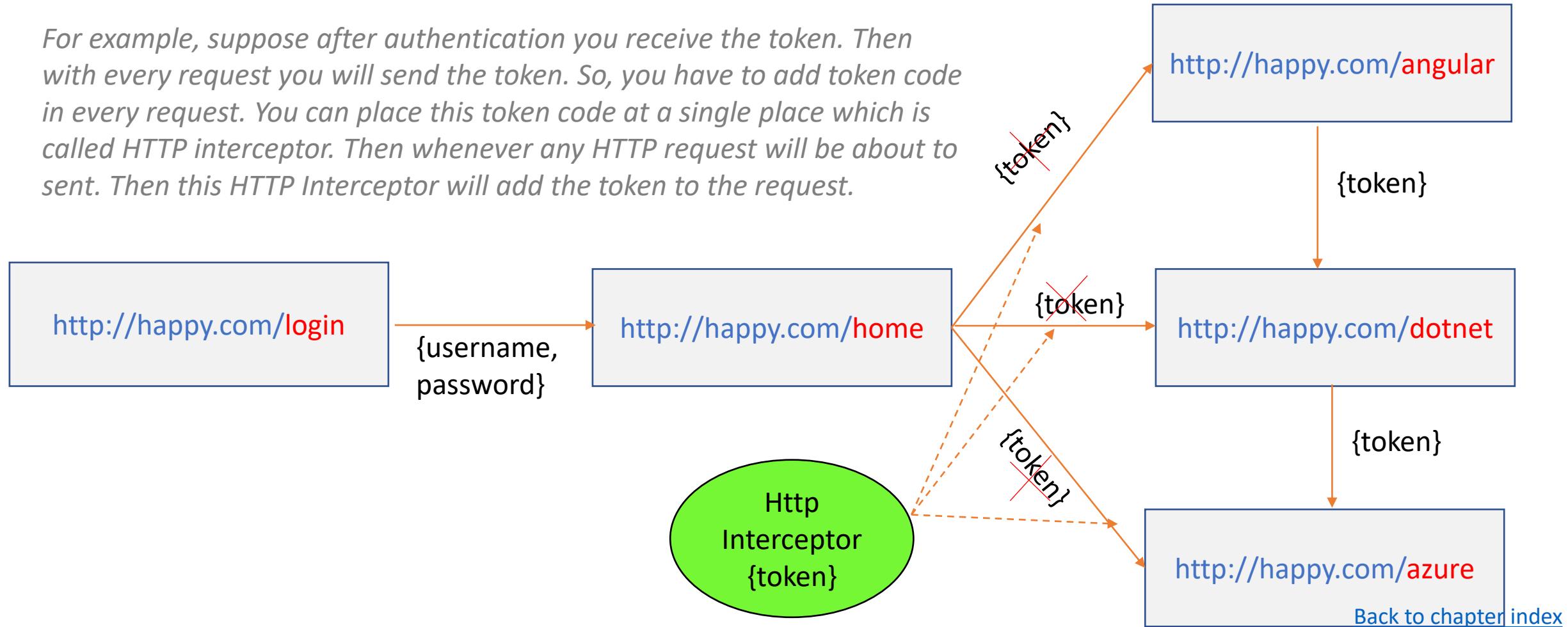
For example, in the image, suppose you are at this login url. Now without entering the username and password, by just changing the url manually, you want to go to the home component. This is not right. So, to prevent this action we use auth guards.

This guard will redirect your request to some other component which you will set in the guard or to the login component only.



- ❖ Interceptor is a special Angular Service, that **can be used to intercept all the request and response calls and modify them to our requirement.**

For example, suppose after authentication you receive the token. Then with every request you will send the token. So, you have to add token code in every request. You can place this token code at a single place which is called *HTTP interceptor*. Then whenever any *HTTP request* will be about to sent. Then this *HTTP Interceptor* will add the token to the request.



```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {

  constructor(private service: AuthService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    //return next.handle(request);
    let token = this.service.GetToken();

    let requestWithToken = request.clone({
      setHeaders: {
        Authorization: 'Bearer' + token
      }
    });

    return next.handle(requestWithToken);
  }
}
```

Suppose your angular is requesting something from the API. But the API is down or there is some network issue. Then you want your Angular App to retry again automatically.

- ❖ The answer is we will use Retry operator of RxJS library.

In image, inside the interceptor if connection fail one time, then Angular will automatically retry 3 more times to make the request.

```
return next.handle(requestWithToken).pipe(  
  retry(3)  
)
```

- ❖ JWT stands for **JSON Web Tokens**.
It has 3 parts:

1. Header
2. Payload
3. Signature



The screenshot shows the jwt.io debugger interface. The "Encoded" section displays the token as a single string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfjoYZgeFONFh7HgQ. The "Decoded" section shows the token split into three parts: HEADER, PAYLOAD, and SIGNATURE. The HEADER contains the algorithm (HS256) and type (JWT). The PAYLOAD contains the subject (1234567890), name (John Doe), and admin status (true). The SIGNATURE is the HMACSHA256 hash of the HEADER and PAYLOAD concatenated with a secret key. A blue button at the bottom right indicates "Signature Verified".

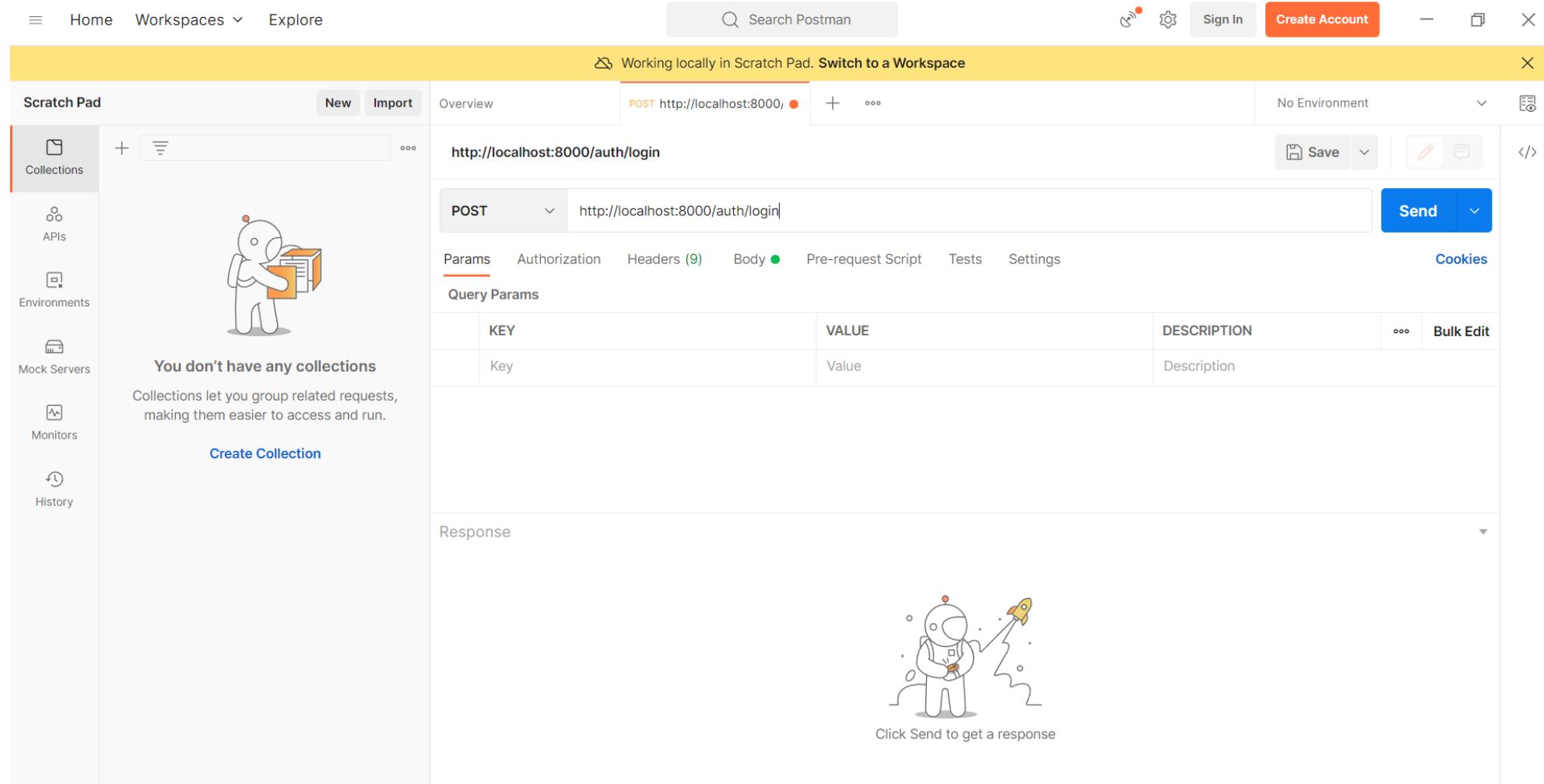
HEADER:
{ "alg": "HS256", "typ": "JWT" }

PAYLOAD:
{ "sub": "1234567890", "name": "John Doe", "admin": true }

VERIFY SIGNATURE
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret) <input type="checkbox"/> secret base64 encoded

Signature Verified

- ❖ Postman is a tool for **testing API's**.

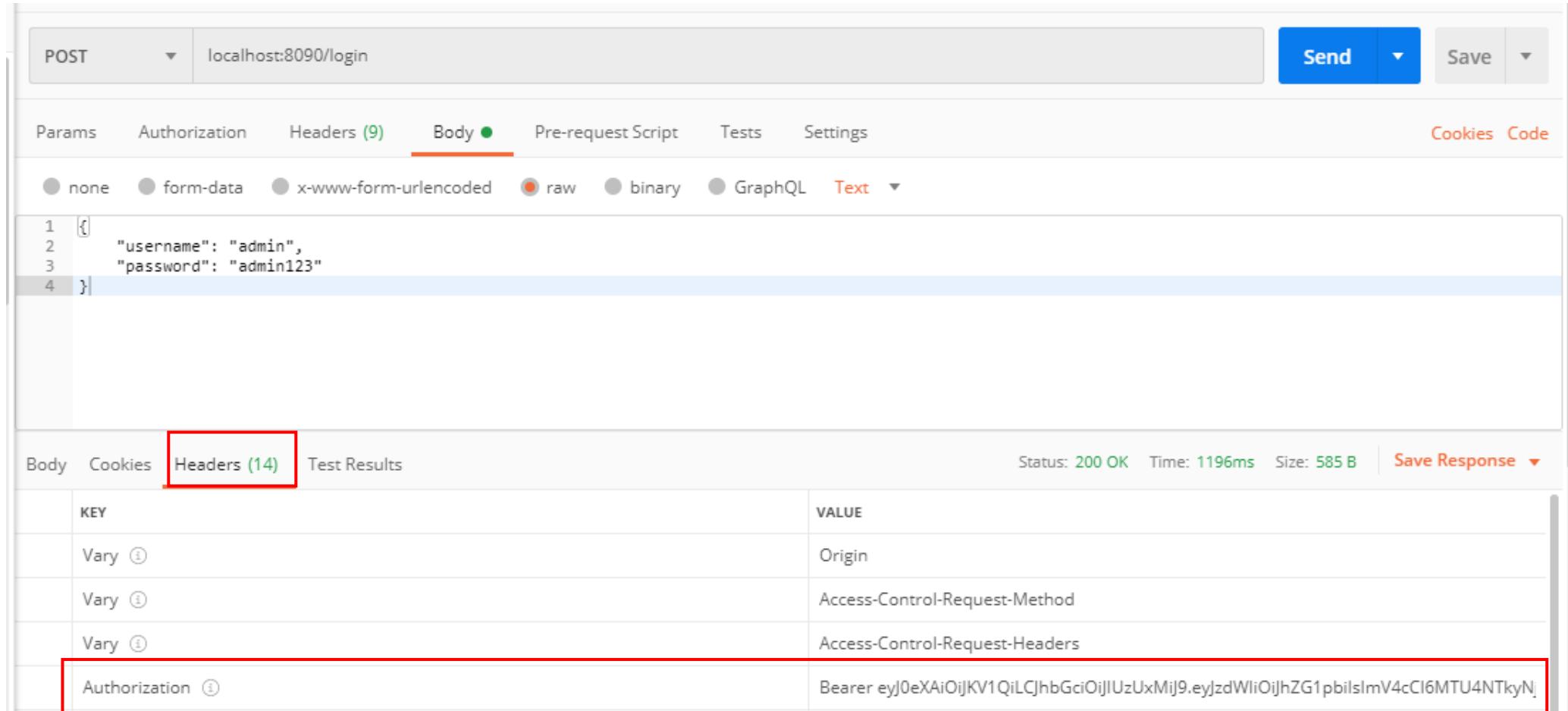


The screenshot shows the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. A yellow banner at the top center says 'Working locally in Scratch Pad. Switch to a Workspace'. The left sidebar has a 'Collections' tab selected, showing a list of icons for 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. A central panel displays a 'Scratch Pad' request for 'http://localhost:8000/auth/login' using a 'POST' method. The 'Headers' tab shows 9 items. The 'Body' tab is selected, showing a green dot icon. Below the body are tabs for 'Params', 'Authorization', 'Tests', and 'Settings'. A 'Cookies' tab is also visible. A 'Query Params' table is present with columns for KEY, VALUE, DESCRIPTION, and Bulk Edit. The table has one row with 'Key' and 'Value' columns. A 'Response' section at the bottom contains a cartoon character holding a rocket and the text 'Click Send to get a response'.

[Back to chapter index](#)

Which part of the request has the token stored when sending to API?

- ❖ Token is sent in the HEADER part of the request.



The screenshot shows the Postman application interface. At the top, it displays a POST request to the URL `localhost:8090/login`. The 'Body' tab is selected, showing the following raw JSON payload:

```
1  {
2    "username": "admin",
3    "password": "admin123"
4 }
```

Below the request, the 'Headers' tab is selected, showing 14 headers. The 'Authorization' header is highlighted with a red box and contains the value: `Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbmlsImV4cCI6MTU4NTkyNj`.

KEY	VALUE
Vary	Origin
Vary	Access-Control-Request-Method
Vary	Access-Control-Request-Headers
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbmlsImV4cCI6MTU4NTkyNj



TOP 100 ANGULAR INTERVIEW QUESTIONS

Chapter 14: Components Communication

Q92. What are the various **ways to communicate** between the components?

Q93. What is **ContentProjection**? What is `<ng-content>`?

Q94. What is **Template Reference Variable** in Angular?

Q95. What is the **role of ViewChild** in Angular?

Q96. How to **access** the child component from parent component with ViewChild?

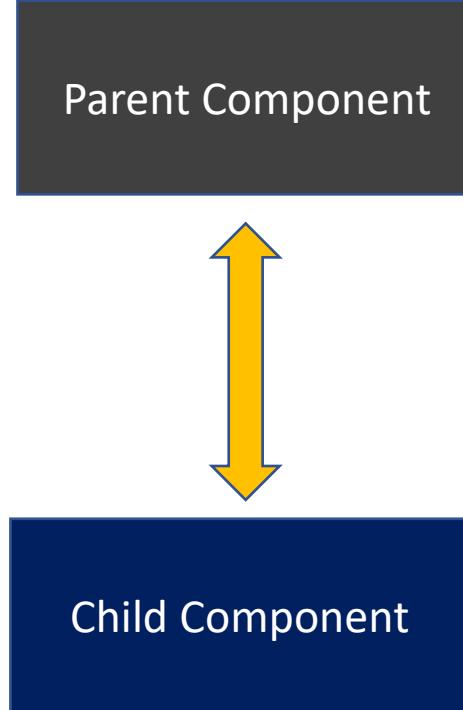
Q97. What is the difference between ViewChild and ViewChildren? What is **QueryList**?

Q98. What is **ContentChild**?

Q99. What is the difference between ContentChild & ContentChildren?

Q100. Compare **ng-Content**, **ViewChild**, **ViewChildren**, **ContentChild** & **ContentChildren**?

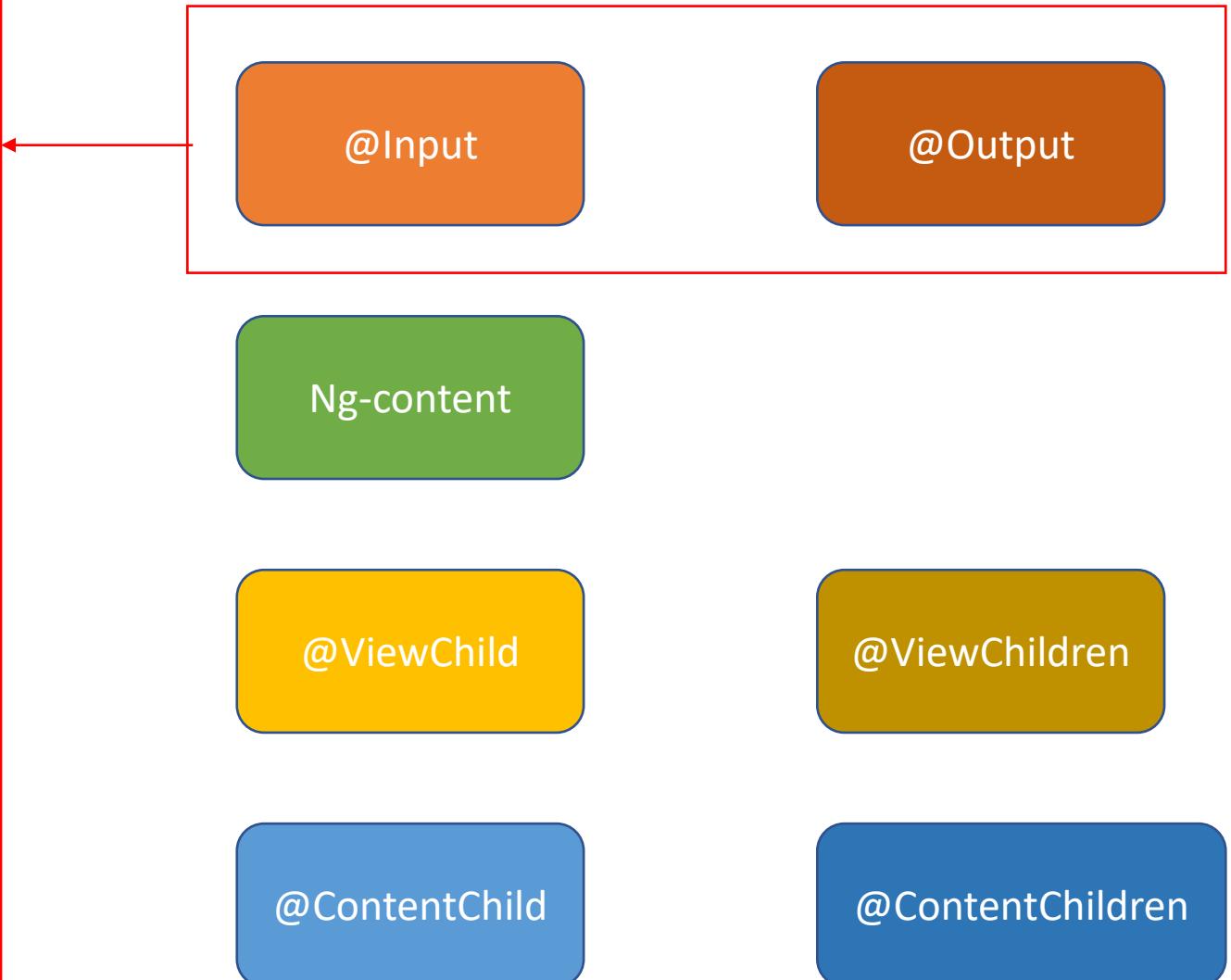
- ❖ When we have multiple components in a single page application. Then many a times we have to send and receive data and content(html) from one component to another component. This is called communication between components.
- ❖ We also call these communicating components as parent and child components.



- ❖ We already discussed `@Input` and `@Output` decorators in previous questions.
- ❖ `@Input` decorator is used to transfer data from parent to child component.
- ❖ `@Output` decorator is used to transfer data from child to parent component.

In upcoming questions we will discuss rest of the ways to transfer content between components.

*Remember `@Input` and `@Output` is used to transfer **data** and rest of decorators are used to transfer content(html).*



- ❖ ng-content is used to insert the dynamic/variable html content from one component to another component.

For example, in the first image we have the <app-sport> selector of the child component in parent component html. And in the second image we have the html of child component app-sport. When the page will be displayed then <ng-content> of title class in child component will be replaced by Football dynamically.

Football

This is one of the top sports.
It is very healthy to play.
Hurreeey...

Play Football

```
<app-sport>
  <h4 class="title">Football</h4>
  <h4 class="footer">Play Football</h4>
</app-sport>
```

```
<div class="card" style="background-color: lightblue;">
  <ng-content select=".title"></ng-content>
  <p>This is one of the top sports.<br>
    It is very healthy to play.<br>
    Hurreeey...<br>
  </p>
  <ng-content select=".footer"></ng-content>
</div>
```

- Template Reference Variable in angular is used to access all the properties of any element inside DOM.

For example, in image #inputBox is the template reference variable of that input element. Then the getInput(inputBox.value) method is used to transfer the value property of the #inputBox. That's how we can send different properties of template reference variable to component class file.

Then in next image, we are receiving the value property in the parameter and printing it via console.log.

```
<h2>Template Reference Variable</h2>

<input type="text" #inputBox>

<br><br>
<button (click)="getInput(inputBox.value)">Click</button>
```

```
export class AppComponent {

  getInput(value: any)
  {
    console.log(value);
  }
}
```

- ❖ @ViewChild decorator is used to access elements of html template file from the component typescript file.

For example, in image you can see emailRef is the template reference variable in html file and it is accessed in component class via @ViewChild decorator.

```
app.component.html M X
src > app > app.component.html > ...
  Go to component
1  <br><br>
2
3  <input #emailRef type="text">
4  <br><br>
5
6  <input type = "password">
7  <br><br>
8
9  <button>Login</button>
10
```

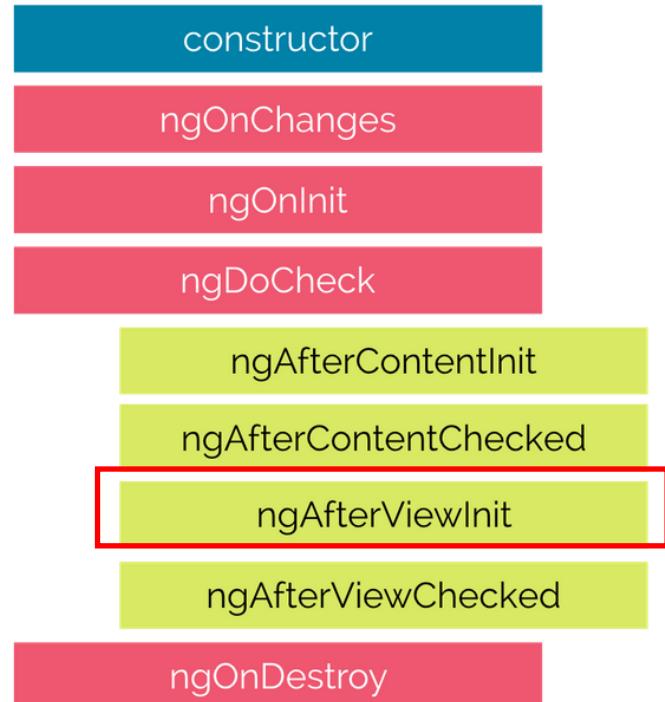
```
app.component.ts M X
src > app > app.component.ts > ...
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9
10    title = 'View-Child';
11
12    @ViewChild('emailRef') email ElementRef: ElementRef | undefined;
13
14 }
```

- ❖ When to use @viewchild?
- 1. When we want to access the element of html component file in the component class file of same component.
- 2. When we want to access the element of child html component file in the parent component class file.

- ❖ To access the child component from parent component, we have to write the code inside the **ngAfterViewInit** lifecycle hook of the parent component class file.

Because if we will write the code in ngOnInit or any other lifecycle hook before ngAfterViewInit, then may be the child component is still not fully developed till then and this may lead to error.

```
export class AppComponent implements AfterViewInit {  
  @ViewChild(ChildComponent, {static:false}) childComponentProp: ChildComponent | undefined;  
  ngAfterViewInit(): void {  
    console.log(this.childComponentProp?.SendToParent());  
  }  
}
```



- ❖ @ViewChildren does the same thing as @ViewChild decorator, but @ViewChild can access only one element, but @ViewChildren can access a **list of elements**.

```
<input #emailRef type="text">
<input #emailRef type="text">
<input #emailRef type="text">
<br><br>
```

```
export class AppComponent implements AfterViewInit {
  title = 'View-Child';
  @ViewChildren('emailRef') emailElementRefChildren: QueryList<ElementRef> | undefined;
  ngAfterViewInit(): void {
    this.emailElementRefChildren?.forEach((item)=>item.nativeElement.value = 100);
  }
}
```

- ❖ QueryList used to manage the **list of elements**.
- ❖ It is used with @ViewChildren to refer the list of elements of type ElementRef.

```
export class AppComponent implements AfterViewInit {  
  title = 'View-Child';  
  
  @ViewChildren('emailRef') emailElementRefChildren: QueryList<ElementRef> | undefined;  
  
  ngAfterViewInit(): void {  
    this.emailElementRefChildren?.forEach(item=>item.nativeElement.value = 100);  
  }  
}
```

- ❖ ContentChild is used to send variable or projected content from one component html to another component class file.
- ❖ To access the element in child component, we have to write the code inside the **ngAfterContentInit** lifecycle hook of the child component class file.



```
<app-child>
  <h3 #refTitle>Football</h3>
</app-child>

@Component({
  selector: 'app-child',
  template: `<h3 #refTitle>Football</h3>`})
class ChildComponent {
  @ContentChild('refTitle') title: ElementRef | undefined;

  ngAfterContentInit(): void {
    console.log(this.title?.nativeElement.innerText);
  }
}
```

- ❖ @ContentChildren does the same thing as @ContentChild decorator, but @ContentChild can access only one element, but @ContentChildren can access a **list of elements**.

```
<app-child>
  <h3 #refTitle>Football</h3>

  <h4 #refTitle>Running</h4>

  <h6 #refTitle>Cricket</h6>
</app-child>
```

```
@ContentChildren('refTitle') titleChildren: QueryList<ElementRef> | undefined;

ngAfterContentInit(): void {
  //For each loop to iterate all the elements in list.
  this.titleChildren?.forEach((item)=> console.log(item.nativeElement.innerText));
}
```

	<ng-content> (Content Projection)	@ViewChild	@ViewChildren	@ContentChild	@ContentChildren
1. Use Case	Used to display variable content from child component html to parent component html.	Used to send content from: 1. Same component html template -> Component class. 2. Child component -> Parent Component class.	Used same as @ViewChild , but in addition it can handle multiple elements .	Used to send projected content from component HTML template file -> Another component class file.	Used same as @ContentChild , but in addition it can handle multiple elements .
2. Example/ Declaration	<code><ng-content select=".title"></ng- content></code>	<code>@ViewChild('emailRef') emailElement: ElementRef</code>	<code>@ViewChildren('emailRe f') emailElement: QueryList<ElementRef></code>	<code>@ContentChild('email Ref') emailElement: ElementRef</code>	<code>@ContentChildren('emal iRef') emailElement: QueryList<ElementRef></code>
3. Template reference variable	No	Yes	Yes	Yes	Yes
4. Type	No type, since it's a html to html content transfer.	ElementRef . (It's a wrapper around an element)	QueryList<ElementRef> . (It's a list of ElementRef)	ElementRef.	QueryList<ElementRef>



All the best for your interviews

Never ever give up



TOP
500

.NET

INTERVIEW QUESTIONS

PART - I

HAPPY RAWAT

ABOUT THE BOOK

Part I - This book contains 250 very important .NET interview questions.

Part II - Next part of this book contains 250 more interview questions.

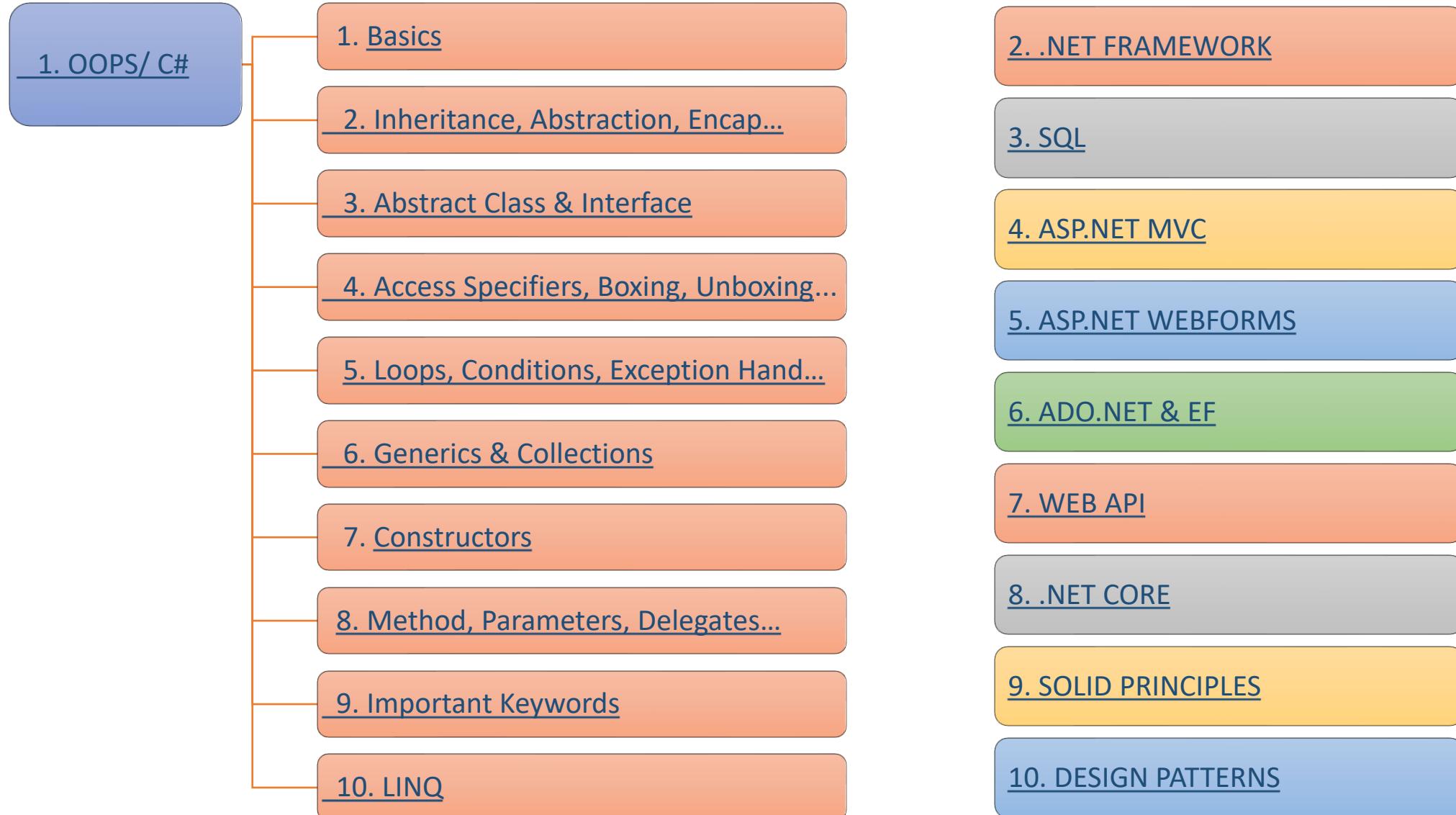


ABOUT THE AUTHOR

Happy Rawat has around 15 years of experience in software development. He helps candidates in clearing technical interview in tech companies.



Chapters



Chapter 1 : OOPS & C# - Basics

Q1. What is C#? What Is the difference between C# and .NET?

Q2. What is OOPS? What are the main concepts of OOPS? V Imp

Q3. What are the advantages of OOPS? V Imp

Q4. What are the limitations of OOPS?

Q5. What are Classes and Objects?

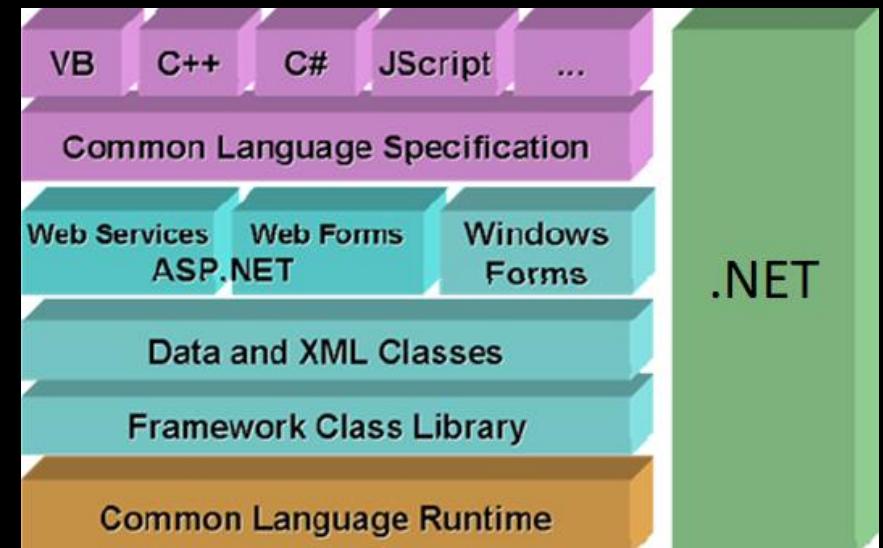
Q6. What are the types of classes in C#?

Q7. Is it possible to prevent object creation of a class in C#?

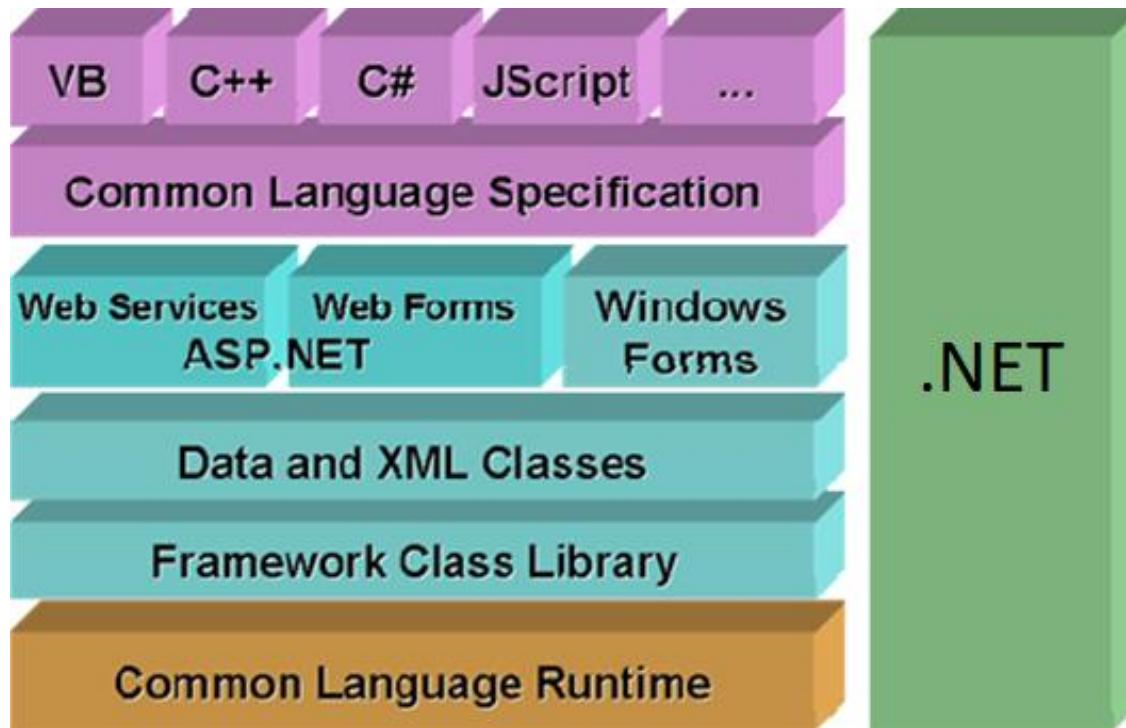
Q8. What is Property?

Q9. What is the difference between Property and Function?

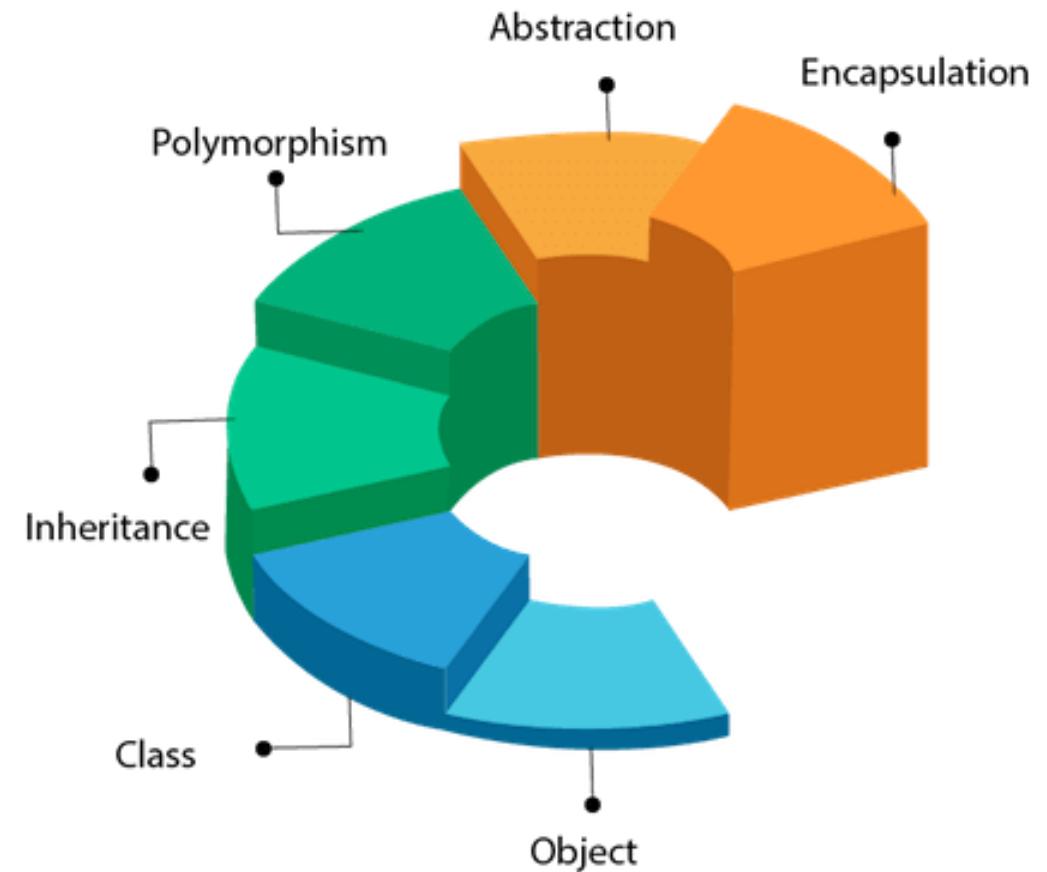
Q10. What are Namespaces?



- ❖ C# is an object-oriented **programming language** which runs on the .NET framework.
- ❖ .NET is a **framework** for building and running software's and applications.

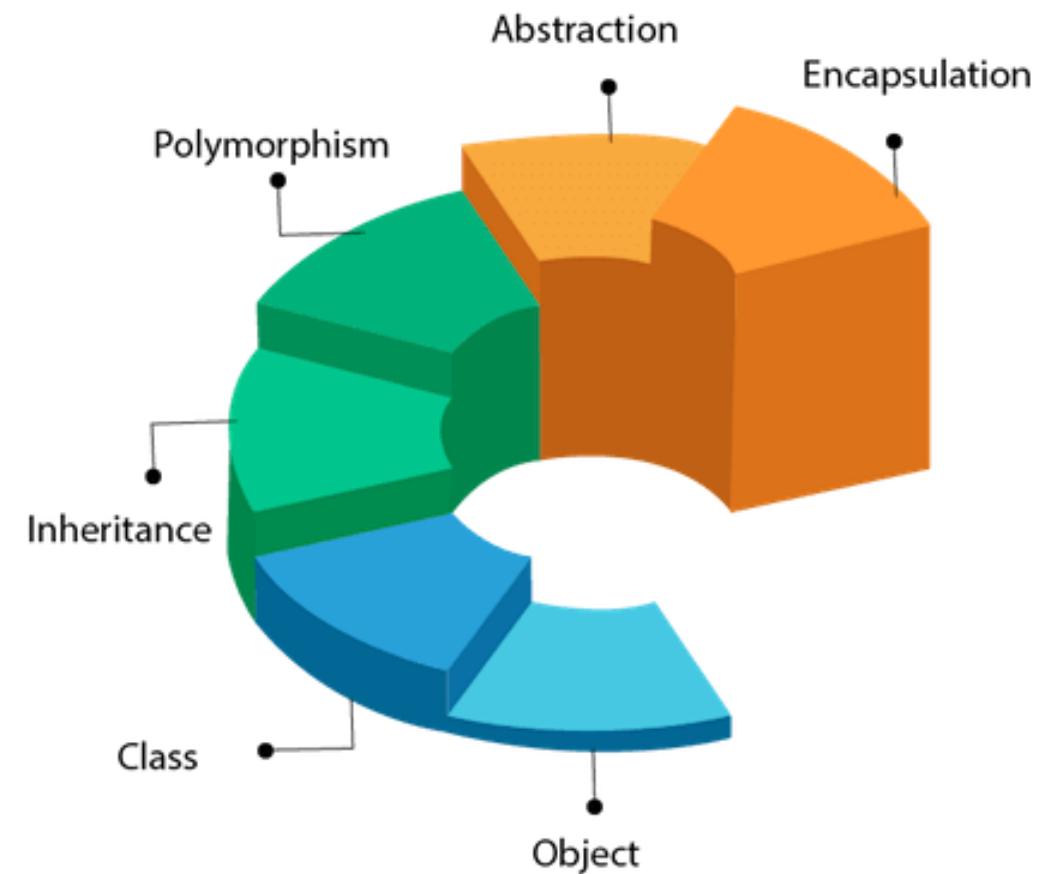


- ❖ OOP stands for **Object-Oriented Programming**, which means it is a way to create software around **objects**.
- ❖ OOPs provide a clear structure for the software's and web applications.



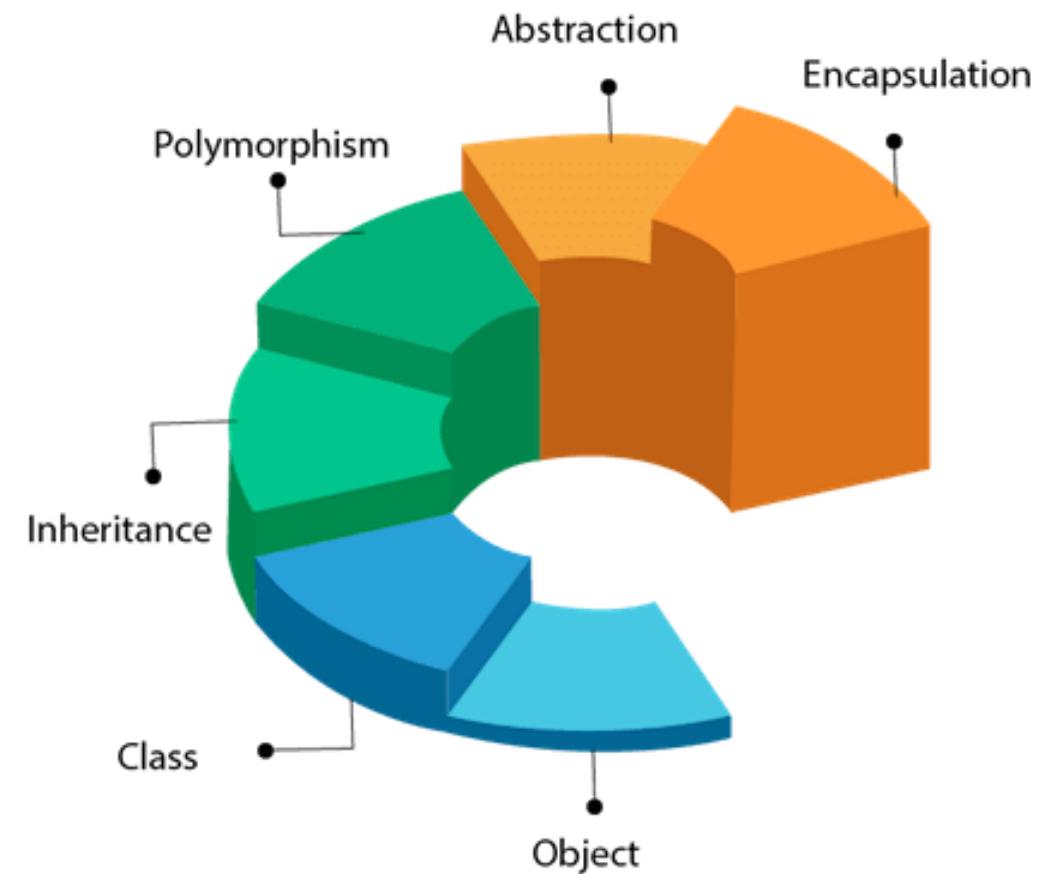
❖ Advantages of OOPS:

1. **Reuse** of code using inheritance.
2. **Flexibility** of code using polymorphism.
3. **Secure** application by using Encapsulation.
4. **Easily scalable** from small to large applications.
5. **Easier troubleshooting** of code because of modularity.



❖ Disadvantage of OOPS:

1. It is not suitable for **small** applications.



- ❖ A class is a **LOGICAL UNIT** or **BLUEPRINT**. It contains fields, methods and properties.

- ❖ Class members are:

1. A **constructor** is a method in the class which gets executed when a class object is created.
2. A **field** is a variable of any type. It is basically the data.
3. A **property** is a member that provides helps in read and write of private field.
4. A **method** is a code block that contains a series of statements.

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

    public void CalculateSalary()
    {
        int salary = Experience * 300000;

        Console.WriteLine(salary);
    }
}
```

Constructor

Field

Property

Method

- ❖ **Object** - An object is an **INSTANCE** of a class.

```
static void Main(string[] args)
{
    Employee objEmployee = new Employee();

    objEmployee.Experience = 3;

    objEmployee.CalculateSalary();

    Console.ReadLine();
}
```

Object

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

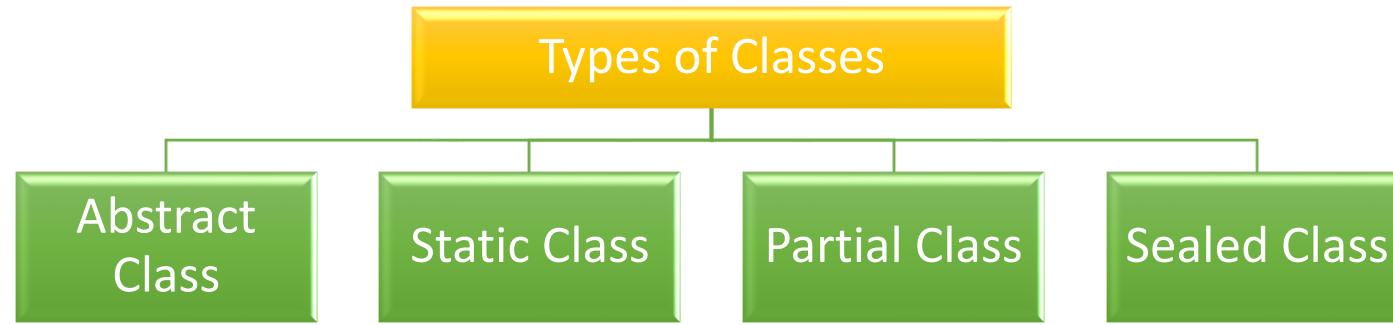
    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

    public void CalculateSalary()
    {
        int salary = Experience * 300000;

        Console.WriteLine(salary);
    }
}
```



```
public abstract class Class1
{
}

public static class Class2
{
}

public partial class Class3
{
}

public sealed class Class4
{}
```

- ❖ Object creation of a class can be prevented by:
 1. Abstract Class
 2. Private Class
 3. Static Class

```
public abstract class Maths
{
}

private class Science
{
}

public static class Social
{
}

public class Student
{
    Maths objMaths = new Maths();

    Science objScience = new Science();

    Social objSocial = new Social();
}
```

- ❖ A property is a class member that provides a flexible mechanism to **read** and **write** private field.

```
public class MyClass
{
    private int myProperty;

    public int MyProperty
    {
        get { return myProperty; }
        set { myProperty = value; }
    }.....

    public string MyProperty1 { get; set; }
}
```

- ❖ Property is a **specialized function** only.
- ❖ Specialized means properties are used only to get and set field values.

```
public class Property_Function
{
    public int MyProperty { get; set; }

    public int MyProperty()
    {
    }
}
```

int Property_Function.MyProperty()

CS0102: The type 'Property_Function' already contains a definition for 'MyProperty'
CS0161: 'Property_Function.MyProperty()': not all code paths return a value
Show potential fixes (Alt+Enter or Ctrl+.)

- ❖ A namespace is a container for a set of related classes and other types.

```
using System.Text;

namespace myNamespace
{
    public class myClass
    {
        StringBuilder sb = new StringBuilder();
    }
}
```

```
namespace MyNamespace
{
    public class MyClass
    {
        // class implementation
    }
}
```

```
using MyNamespace;
MyClass myInstance = new MyClass();
```

Chapter 2 : OOPS & C# - Inheritance, Abstraction, Encapsulation & Polymorphism

Q11. What is Inheritance? When to use Inheritance?

Q12. What are the different types of Inheritance? V Imp

Q13. Does C# support Multiple Inheritance?

Q14. How to prevent a class from being Inherited?

Q15. Are private class members inherited to the derived class?

Q16. What is Abstraction? How to implement abstraction? V Imp

Q17. What is Encapsulation? How to implement encapsulation? V Imp

Q18. What is the difference between Abstraction and Encapsulation?

Q19. What is Polymorphism and what are its types? When to use polymorphism?

Q20. What is Method Overloading? In how many ways a method can be overloaded?

Q21. When should you use method overloading in real applications?

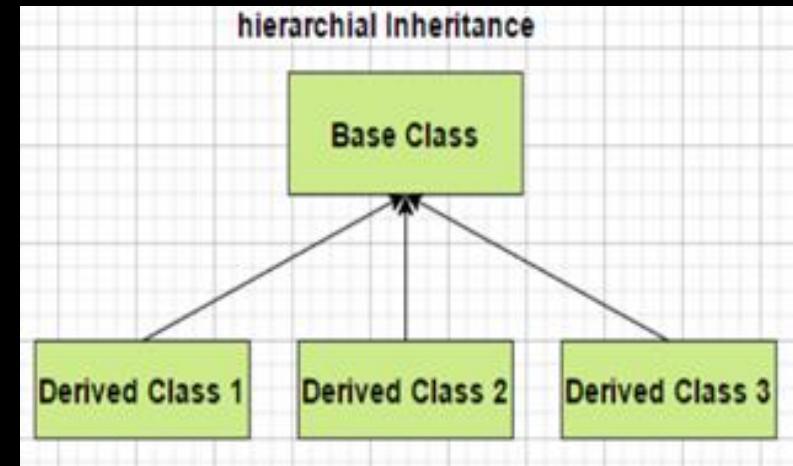
Q22. If two methods are same except return type, then methods are overloaded or what will happen?

Q23. What is the difference between Overloading and Overriding? V Imp

Q24. What is the use of Overriding?

Q25. If a method is marked as virtual, do we must have to "override" it from the child class?

Q26. What is the difference between Method Overriding and Method Hiding?



- Inheritance is creating a **PARENT-CHILD** relationship between two classes, where child class will **automatically** get the properties and methods of the parent.
- Inheritance is good for: **REUSABILITY** and **ABSTRACTION** of code

```
public class ContractEmployee : Employee
{
    //No method or Property here
}
```

```
public class Employee
{
    public int Experience { get; set; }

    public void CalculateSalary()
    {
        int salary = Experience * 300000;

        Console.WriteLine("salary:{0} ", salary);
    }
}

public class PermanentEmployee : Employee
{
    //No method or Property here
}

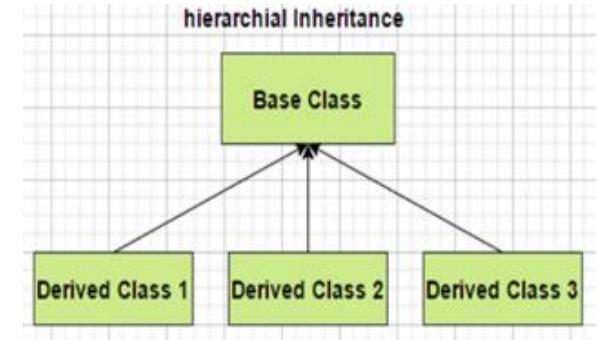
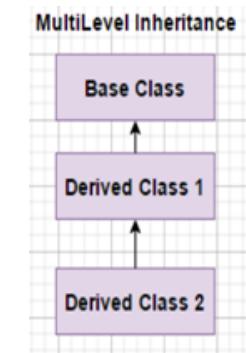
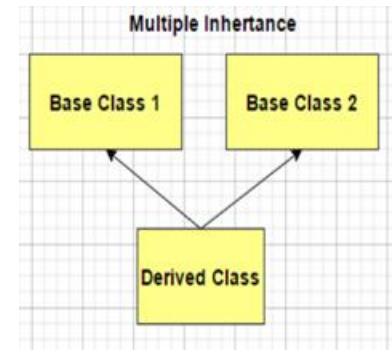
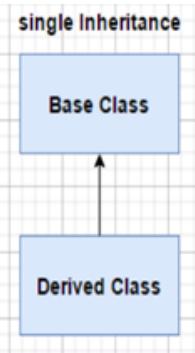
static void Main(string[] args)
{
    PermanentEmployee pEmployee = new PermanentEmployee();
    pEmployee.Experience = 5;
    pEmployee.CalculateSalary();

    Console.ReadLine();
}
```

Base/ Parent/ Super class

Derived/ Child/ Sub class

CalculateSalary() method is not present in PermanentEmployee class, but it will get it automatically from it's parent



```
class BaseClass1
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
class DerivedClass1 : BaseClass1
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
```

```
class BaseClass2
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
interface I2
{
    void Fly();
}
class DerivedClass2 : BaseClass2, I2
{
    public void Eagle()
    {
        Console.WriteLine("Eagle");
    }
    public void Fly()
    {
        Console.WriteLine("Fly");
    }
}
```

```
class BaseClass3
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
class DerivedClass3 : BaseClass2
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
class DerivedClass4 : DerivedClass3
{
    public void Labrador()
    {
        Console.WriteLine("Labrador");
    }
}
```

```
class BaseClass4
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
class DerivedClass5 : BaseClass4
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
class DerivedClass6 : BaseClass4
{
    public void Cat()
    {
        Console.WriteLine("Cat");
    }
}
```

- ❖ NO
- ❖ C# support inheritance by using **multiple Interfaces**. This is an alternative way of multiple inheritance.

- ❖ By using **SEALED** keyword in class

```
public sealed class Employee
{
    public void GetSalary()
    {
        Console.WriteLine("100000");
    }
}
public class PermanentEmployee : Employee
```

CS0509: 'PermanentEmployee': cannot derive from sealed type 'Employee'

- ❖ By using **STATIC** keyword in base class

```
public static class Employee
{
    public static void GetSalary()
    {
        Console.WriteLine("100000");
    }
}
public class PermanentEmployee : Employee
```

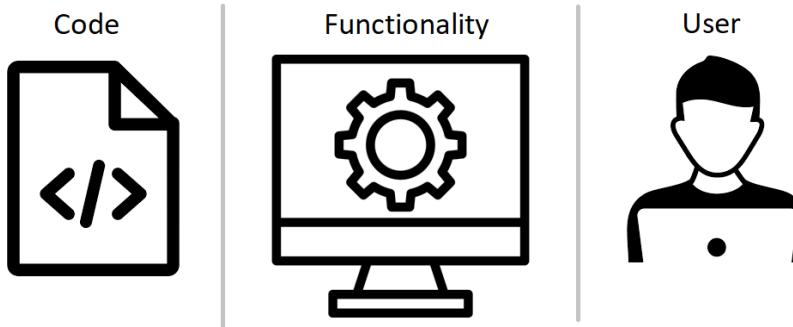
CS0709: 'PermanentEmployee': cannot derive from static class 'Employee'

- ❖ Difference between sealed & static is, you can create the object of sealed class, but you cannot create the object of static class.



- ❖ No, the private members cannot be inherited to derived class.
- ❖ Only public and protected class members can be inherited.

- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.



```
Employee employee = new Employee();

int sal = employee.CalculateSalary();

Console.WriteLine( sal );

//Output: 3000000
```

```
String name = "InterviewHappy";

string firstname = name.Substring(8);

Console.WriteLine(firstname);

//Output: Interview
```

Background/Hidden method.

```
public abstract class EmployeeSalary
{
    public int CalculateSalary()
    {
        return 10 * 300000;
    }
}
```

```
public class Employee : EmployeeSalary
{
}
```

- ❖ Mostly we use **abstract classes** and **interfaces** for implementing abstraction.

- ❖ Encapsulation means **WRAPPING** of data and methods into a single unit.

```
public class Employee
{
    public int empExperience;
}

static void Main(string[] args)
{
    Employee objEmployee = new Employee();
    objEmployee.empExperience = 3;
}
```

Field/Data is accessed without property or function. It will work but it violating encapsulation

```
public class Employee
{
    //Make field private
    private int empExperience;

    public int EmpExperience
    {
        get { return empExperience; }
        set { empExperience = value; }
    }

    //Shortcut Property
    //public int EmpExperience { get; set; }
}
```

This field cannot be accessed from outside without the property

```
static void Main(string[] args)
{
    Employee objEmployee = new Employee();
    objEmployee.EmpExperience = 3;
}
```

- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.
- ❖ Encapsulation means **WRAPPING** of data and methods into a single unit.

```
public abstract class EmployeeSalary
{
    public int CalculateSalary()
    {
        return 10 * 300000;
    }
}

public class Employee : EmployeeSalary
{}
```

```
Employee employee = new Employee();

int sal = employee.CalculateSalary();

Console.WriteLine( sal );

//Output: 3000000
```

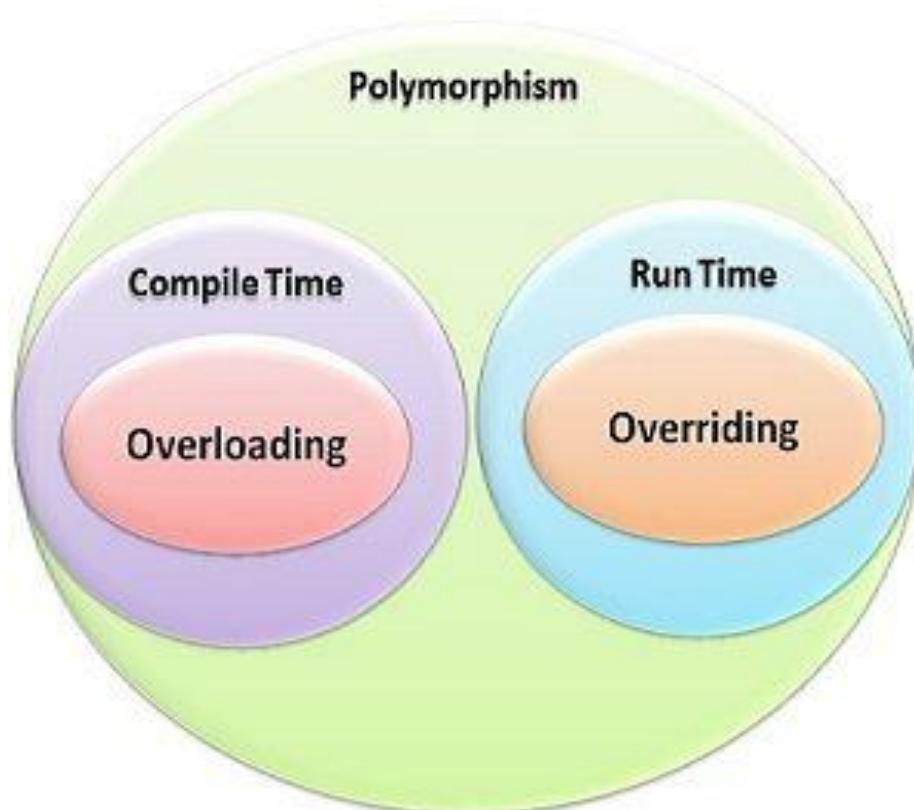
```
public class Employee
{
    //Make field private
    private int empExperience;

    public int EmpExperience
    {
        get { return empExperience; }

        set { empExperience = value; }
    }

    //Shortcut Property
    //public int EmpExperience { get; set; }
}
```

- ❖ Polymorphism is the ability of a variable, object, or function to take on **MULTIPLE FORMS**.



o use polymorphism?

```
public class Polymorphism
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

Function with
same name but
have multiple
forms

```
public string Add(string str1, string str2)
{
    return str1 + str2;
}
```

```
Polymorphism obj = new Polymorphism();

int i = obj.Add(50, 60);

string str = obj.Add("Interview", "Happy");

Console.WriteLine(i + " - " + str);

//Output: 110 - InterviewHappy
```

- Method overloading is a type of polymorphism in which we can create multiple methods of the **same name in the same class**, and all methods work in different ways.

```
public class MethodOverloading
{
    public int Add(int a, int b)
    {
        return a + b;
    }

    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }

    public double Add(double a, double b, int c)
    {
        return a + b + c;
    }

    public double Add(double a, int c, double b)
    {
        return a + b + c;
    }
}
```

1. Number of parameters are different

2. Type of parameters are different

3. Order of parameters is different

```
name.Substring(b.ToString().Length);
```

▲ 1 of 2 ▼ string string.Substring(int startIndex)

Retrieves a substring from this instance. The substring starts at a specified **startIndex**: The zero-based starting character position of a substring in this

```
public class MethodOverloading
{
    public int Add(int a, int b)
    {
        return a + b;
    }

    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }

    public double Add(double a, double b, int c)
    {
        return a + b + c;
    }

    public double Add(double a, int c, double b)
    {
        return a + b + c;
    }
}
```

- ❖ No, this will show compile time error.

```
public class Employee
{
    public int GetSalary(int designation)
    {
        return 100000;
    }

    public string GetSalary(int designation)
    {
        string Employee.GetSalary(int designation)
        return "100000";
    }
}
```

CS0111: Type 'Program.Employee' already defines a member called 'GetSalary' with the same parameter types
CA1822: Member 'GetSalary' does not access instance data and can be marked as static
Show potential fixes (Alt+Enter or Ctrl+.)

❖ Method Overriding

1. Multiple methods of same name are in **different class**.
2. **Inheritance is used**, as it is in different class.
3. Both methods have **same signature**.
4. It's a **run time** polymorphism.
5. **Virtual & override** keywords.

```
? public class BaseClass
{
    public virtual void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public override void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}
```

```
static void Main(string[] args)
{
    DerivedClass objDerived = new DerivedClass();
    objDerived.Greetings();
    Console.ReadLine();
}

//Output: DerivedClass Hello
```

❖ Method Overloading

1. Multiple methods of same name in **single class**.
2. No need of **inheritance**, as it is in single class.
3. All methods have **different signature**.
4. It's a **compile time** polymorphism.
5. No special keyword used.

❖ Method Overriding

1. Multiple methods of same name in **different class**.
2. **Inheritance is used**, as it is in different class.
3. All methods have **same signature**.
4. It's a **run time** polymorphism.
5. **Virtual & override** keywords.

- ❖ Overriding is used to modify and provide a new implementation of the method inherited from a base class.

```
public class Testing : Technology
{
    public override void TechnicalSkill()
    {
        Console.WriteLine("Testing");
    }
}
```

```
static void Main(string[] args)
{
    Testing test = new Testing();
    test.TechncialSkill();
    test.CommunicationSkill();
}

//Output: Testing English
```

```
public class Technology
{
    public virtual void TechnicalSkill()
    {
        Console.WriteLine("Coding");
    }
    public virtual void CommunicationSkill()
    {
        Console.WriteLine("English");
    }
}
```

```
public class Java : Technology
{
}

public class DotNet : Technology
{
}
```

- ❖ NO. Overriding virtual method is optional.

```
FROM THE CHILD CLASS:  
public class Technology  
{  
    public virtual void TechnicalSkill()  
    {  
        Console.WriteLine("Coding");  
    }  
}
```

```
public class Java : Technology  
{  
}  
  
static void Main(string[] args)  
{  
    Java java = new Java();  
    java.TechanicalSkill();  
}  
  
//Output: Coding
```

- ❖ In Method Hiding, you can completely hide the implementation of the methods of a base class from the derived class using the **new keyword**.

```
public class BaseClass
{
    public virtual void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public override void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}

static void Main(string[] args)
{
    BaseClass objDerived = new DerivedClass();

    objDerived.Greetings();
}

//Output: DerivedClass Hello
```

```
public class BaseClass
{
    public void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public new void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}
```

```
static void Main(string[] args)
{
    BaseClass objDerived = new DerivedClass();

    objDerived.Greetings();
    Console.ReadLine();
}

//Output: BaseClass Hello
```

Chapter 3 : OOPS & C# - Abstract Class & Interface

Q27. What is the difference between **Abstract** class and an **Interface**? V Imp

Q28. When to **use Interface** and when Abstract class in real applications? V Imp

Q29. Why to **create Interfaces** in real applications?

Q30. Can we define body of Interfaces methods? When to define methods in Interfaces?

Q31. Can you create an instance of an Abstract class or an Interface?

Q32. Do Interface can have a **Constructor**?

Q33. Do abstract class have Constructors in C#?

Q34. What is the difference between abstraction and abstract class? V Imp

Q35. Can Abstract class be Sealed or Static in C#?

Q36. Can you declare **abstract methods** as private in C#?

Q37. Does Abstract class support multiple Inheritance?



1. Abstract class contains both **DECLARATION** & **DEFINITION** of methods.

```
public abstract class Employee
{
    public abstract void Project();
    public void Role()
    {
        Console.WriteLine("Engineer");
    }
}
```

Method Declared

Method Defined

1. Interface should contain **DECLARATION** of methods.

❖ *With C# 8.0, you can now have default implementations/definition of methods in an interface. But that is recommended in special case*.*

```
interface IEmployee
{
    public void Project1();
    public void Manager1();
}
```

Only method
Declaration is
allowed

2. Abstract class keyword: **ABSTRACT**

2. Interface keyword: **INTERFACE**

3. Abstract class does not support **multiple inheritance**

```
public abstract class Employee
{
    public abstract void Project();

    public void Role()
    {
        Console.WriteLine("Engineer");
    }
}

public abstract class Employee1
{
    public abstract void Project1();

    public void Role1()
    {
        Console.WriteLine("Engineer1");
    }
}

public class PermanentEmployee: Employee, Employee1
{}
```

3. Interface supports **multiple inheritance**.

```
interface IEmployee1
{
    public void Project1();
}

interface IEmployee2
{
    public void Project2();
}

public class NewEmployee : IEmployee1, IEmployee2
{
    public void Project1()
    {
        Console.WriteLine("Print 1");
    }

    public void Project2()
    {
        Console.WriteLine("Print 2");
    }
}
```

4. Abstract class can have **constructors**.

```
public abstract class Employee1
{
    public Employee1()
    {
    }

    public abstract void Project1();

    public void Role1()
    {
        Console.WriteLine("Engineer1");
    }
}
```

4. Interface do not have constructors.

```
interface IEmployee1
{
    public IEmployee1()
    {
    }

    public void Project1();
}
```

Abstract Class	Interface
1. Abstract class contains both DECLARATION & DEFINITION of methods.	Mostly Interfaces contain DECLARATION of methods. From C# 8.0 definition is also possible.
2. Abstract class keyword: ABSTRACT	2. Interface keyword: INTERFACE
3. Abstract class does not support multiple inheritance	3. Interface supports multiple inheritance .
4. Abstract class can have constructors .	4. Interface do not have constructors.

❖ When to use Interface?

An interface is a good choice when you know a method has to be there, but it can be implemented **DIFFERENTLY** by independent derived classes.

```
15: public class PermanentEmployee
{
}
public class ContractualEmployee
{
}
```

```
interface IEmployee
{
    public void AssignEmail();
    public void AssignManager();
}
```

❖ When to use Abstract class?

Abstract class is a good choice when you are sure some methods are concrete/defined and must be implemented in the **SAME WAY** in all derived classes.

❖ Normally we prefer Interface because it gives us the flexibility to modify the behavior at later stage.

```
public class PermanentEmployee
```

```
{
```

```
public class ContractualEmployee
```

```
{
```

```
public abstract class EmployeeDress
```

```
{
```

```
    public abstract void DressCode();
```

```
    public void DressColor()
```

```
{
```

```
    Console.WriteLine("BLUE");
```

```
}
```

```
}
```



```
public class PermanentEmployee
{
}

public class ContractualEmployee
{
}
```

❖ Benefits of Interfaces:

1. Help in defining the **contract** of the system.
2. **Unit testing** is easy in application having interfaces.
3. Used for implementing **dependency injection**.

```
interface IEmployee
{
    public void AssignEmail();
}
```

- ❖ Yes. From C# 8.0 it is possible to put method bodies in Interfaces.

```
public interface IEmployee
{
    public void Role()
    {
        Console.WriteLine("Developer");
    }
}
```

- ❖ No. Abstract class and interface purpose is to act as base class via inheritance. There object creation is not possible.

```
public abstract class Employee
{
    public abstract int Salary();

    public string Role()
    {
        return "Developer";
    }
}
```

```
public interface IEmployee
{
    public int Role();
}
```

```
static void Main(string[] args)
{
    Employee employee = new Employee();

    IEmployee employee2 = new IEmployee();
}
```

interface Can_you_create_object_of_an_abstract_class.Program.IEmployee

CS0144: Cannot create an instance of the abstract type or interface 'Program.IEmployee'

Show potential fixes (Alt+Enter or Ctrl+.)

[back to chapter index](#)

- ❖ **NO.** Interface can only be used for inheritance, so constructor is not required.

```
interface IEmployee1
{
    public IEmployee1()
    {
    }
    public void Project1();
}
```

IEmployee1()

CS0526: Interfaces cannot contain instance constructors

- ❖ YES, Abstract class can have constructors.
- ❖ The reason is, when the abstract class is inherited in the derived class, and whenever the object of the derived class is created then FIRST the constructor of the abstract/ base class will be called and then only the constructor of derived class will be called.

```
public abstract class Employee
{
    public Employee()
    {
        Console.WriteLine("Employee Constructor");
    }
}

public class PermanentEmployee : Employee
{
}
```

```
static void Main(string[] args)
{
    //Employee employee = new Employee();
    PermanentEmployee employee = new PermanentEmployee();
}

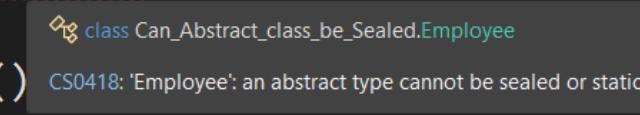
//Output: Employee Constructor
```

- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.
- ❖ Abstract class is one of the ways to implement abstraction.

- ❖ NO. Abstract class are used for inheritance, but sealed and static both will not allow the class to be inherited.

```
public sealed abstract class Employee
{
    public abstract int Salary();

    public string Role()
    {
        return "Developer";
    }
}
```



The screenshot shows a C# code editor with the following code:

```
public sealed abstract class Employee
{
    public abstract int Salary();

    public string Role()
    {
        return "Developer";
    }
}
```

A tooltip is displayed over the word "Employee" in the first line, showing the class name and the error message: "CS0418: 'Employee': an abstract type cannot be sealed or static".

- ❖ NO. Abstract methods are only declaration, and they must be implemented in the derive class, therefore they can not be private.

```
public abstract class Employee
{
    private abstract int Salary();
    public string Role()
    {
        return "Developer";
    }
}
```

int Employee.Salary()
CS0621: 'Employee.Salary()': virtual or abstract members cannot be private

- ❖ NO. Only at maximum one abstract class can be used for inheritance.
- ❖ But multiple interfaces can be used.

```
public abstract class Technology
{
    public abstract void DotNet();
}

public abstract class Role
{
    public abstract void Lead();
}

public class Employee : Technology, Role
```

class Does_Abstract_class_support_multiple_Inheritance.Role

CS1721: Class 'Employee' cannot have multiple base classes: 'Technology' and 'Role'

Chapter 4 : OOPS & C# - Access Specifiers, Boxing, Unboxing, String & StringBuilder

Q38. What are Access Specifiers? V Imp

Q39. What is internal access modifier? Show example?

Q40. What is the default access modifier in a class?

Q41. What is Boxing and Unboxing? V Imp

Q42. Which one is explicit Boxing or Unboxing?

Q43. Is Boxing and Unboxing good for performance?

Q44. What are the basic string operations in C#?

Q45. What is the difference between “String” and “StringBuilder”? V Imp

Q46. When to use String and when StringBuilder in real applications?

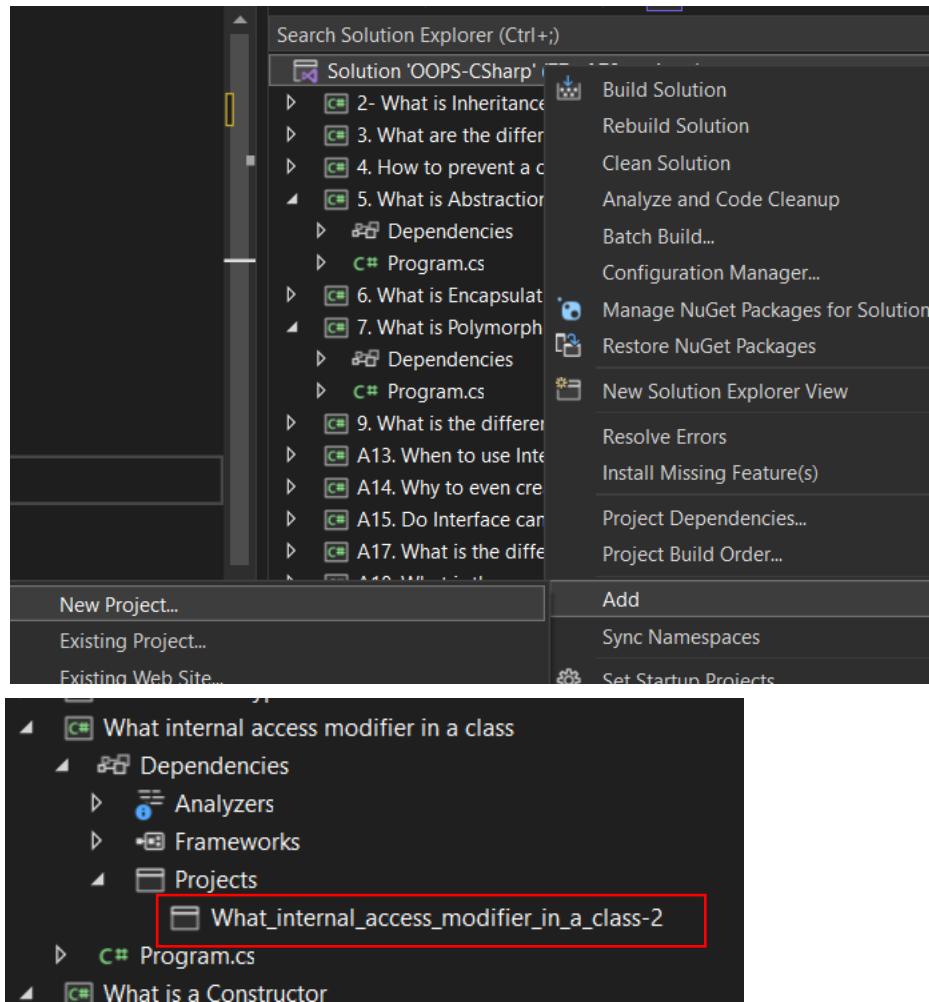
Q47. What is String Interpolation in C#?

Access Specifier	Inside Same Assembly where member is declared			Other Assembly where containing Assembly is referenced	
	Inside Same Class	Inside Derived Class	Other Code	Inside Derived Class	Other Code
Public	✓	✓	✓	✓	✓
Private	✓	✗	✗	✗	✗
Internal	✓	✓	✓	✗	✗
Protected	✓	✓	✗	✓	✗
ProtectedInternal	✓	✓	✓	✓	✗

- ❖ Access specifiers are keywords to specify the **accessibility** of a class, method, property, field.
- ❖ The keywords are – Public, Private, Protected, Internal, Protected Internal.

Access Specifier	Inside Same Assembly where member is declared			Other Assembly where containing Assembly is referenced	
	Inside Same Class	Inside Derived Class	Other Code	Inside Derived Class	Other Code
Public	✓	✓	✓	✓	✓
Private	✓	✗	✗	✗	✗
Internal	✓	✓	✓	✗	✗
Protected	✓	✓	✗	✓	✗
ProtectedInternal	✓	✓	✓	✓	✗

- Internal modifier is used to tell that access is limited to the current assembly.



```
namespace What_internal_access_modifier_in_a_class_2
{
    internal class Employee
    {
        public int GetSalary()
        {
            return 100000;
        }
    }
}
```

```
using What_internal_access_modifier_in_a_class_2;

namespace What_internal_access_modifier_in_a_class
{
    public class Program
    {
        static void Main(string[] args)
        {
            Employee employee = new Employee();
        }
    }
}
```

CS0122: 'Employee' is inaccessible due to its protection level

Show potential fixes (Alt+Enter or Ctrl+.)

Internal is the default access modifier of a class.

```
??? class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

- ❖ **Boxing** - Boxing is the process of converting from value type to reference type.
- ❖ **Unboxing** - Unboxing is the process of converting reference type to value type.

```
static void Main(string[] args)
{
    int num = 100;

    object obj = num;      //Boxing

    int i = (int)obj;      //Unboxing

}
```

- ❖ We internally use boxing when item is added to ArrayList. And we use unboxing when item is extracted from ArrayList.

```
int num = 100;  
ArrayList arrayList = new ArrayList();  
arrayList.Add(i); //Boxing  
  
int k = (int)arrayList[0]; //Unboxing
```

- ❖ Unboxing is explicit conversion process.

```
static void Main(string[] args)
{
    int num = 100;

    object obj = num;      //Boxing

    int i = (int)obj;      //Unboxing

}
```

- ❖ No, it is recommended to use boxing and unboxing when it is necessary only.
- ❖ Converting from one type to another type is not good from performance point of view.

```
ArrayList arrayList = new ArrayList();
arrayList.Add(i);           //Boxing
int k = (int)arrayList[0]; //Unboxing
```

❖ **Concatenate:**

Two strings can be concatenated either by using a System.String.Concat or by using + operator.

```
string str1 = "This is one";
string str2 = "This is two";
string str2 = str1 + str2;

//Output: This is one This is two
```

❖ **Replace:**

Replace(a,b) is used to replace a string with another string.

```
string str1 = "This is one";
string str2 = str1.Replace("one", "two");

//Output: This is two
```

❖ **Trim:**

Trim() is used to trim the white spaces in a string at the end.

```
string str1 = "This is one  ";
str1.Trim();
//Output: "This is one"
```

❖ **Contains:**

Check if a string contains a pattern of substring or not.

```
string str = "This is test";
bool result = str.Contains("test");
```

//Output: true

- ❖ String is **IMMUTABLE** in C#.

It means if you defined one string then you couldn't modify it. Every time you will assign some value to it, it will create a new string.

```
String str1 = "Interview";  
str1 = str1 + "Happy";  
Console.WriteLine(str1);
```

Both these strings are different and occupy different memory in process

- ❖ StringBuilder is **MUTABLE** in C#.

This means if any manipulation will be done on string, then it will not create a new instance every time.

```
StringBuilder str2 = new StringBuilder();  
str2.Append("Interview");  
str2.Append("Happy");
```

Only one string in memory.

- ❖ If you want to change a string **multiple times**, then **StringBuilder** is a better option from **performance** point of view because it will not require new memory every time.

```
String str1 = "Interview";  
  
str1 = str1 + "Happy";  
  
Console.WriteLine(str1);
```

```
StringBuilder str2 = new StringBuilder();  
  
str2.Append("Interview");  
  
str2.Append("Happy");
```

- ❖ String interpolation is a technique that enables you to insert expression values into strings.

```
static void Main(string[] args)
{
    string name = "Happy";

    Console.WriteLine("My name is {0}", name);

    Console.WriteLine("My name is " + name);

    //String Interpolation
    Console.WriteLine($"My name is {name}");

    //Output: My name is Happy
}
```

Chapter 5 : OOPS & C# - Loops, Conditions, Exception Handling

Q48. What are the **Loop types** in C#? When to use what in real applications?

Q49. What is the difference between “**continue**” and “**break**” statement? V Imp

Q50. What are the alternative ways of writing if-else conditions? When to use what?

Q51. How to implement **Exception Handling** in C#? V Imp

Q52. Can we execute multiple Catch blocks?

Q53. When to use Finally in real applications?

Q54. Can we have only “**Try**” block without “**Catch**” block in real applications?

Q55. What is the difference between **Finally** and **Finalize**?

Q56. What is the difference between “**throw ex**” and “**throw**”? Which one to use in real applications? V Imp

```
//While Loop  
  
int i = 0; -----> Initialize  
while(i < 5) -----> Condition  
{  
    Console.WriteLine(i);  
    i++; -----> Increment  
}  
  
//Output: 0 1 2 3 4
```

```
//Do While  
  
int j = 100;  
do  
{  
    Console.WriteLine(j);  
    j++;  
}  
while(j < 10);  
  
//Output: 100
```

Whether condition true or false, this statement will run at least first time.

```
//For Loop  
  
for(int k = 0 ; k < 5; k++)  
{  
    Console.WriteLine(k);  
}  
  
//Output: 0 1 2 3 4
```

```
//ForEach Loop  
  
int[] arr = new int[] { 1,2,3,4};  
  
foreach (int items in arr)  
{  
    Console.WriteLine(items);  
}  
  
//Output: 1 2 3 4
```

- ❖ **Continue** statement end the loop iteration and transfer the control to the beginning of the loop.
- ❖ **Break** statement end the loop iteration and exit the loop.

```
for (int i = 0; i < 5; i++)  
{  
    if (i == 3)  
    {  
        continue;  
    }  
    Console.WriteLine("Print: " + i);  
}
```

//Output:
Print: 0
Print: 1
Print: 2
Print: 4

```
for (int i = 0; i < 5; i++)  
{  
    if (i == 3)  
    {  
        break;  
    }  
    Console.WriteLine("Print: " + i);  
}
```

//Output:
Print: 0
Print: 1
Print: 2

❖ Switch statement

```
switch (level)
{
    case "SE":
        salary = 70000;
        break;
    case "SSE":
        salary = 100000;
        break;
    default:
        salary = 150000;
        break;
}

Console.WriteLine(salary);
```

❖ Ternary Operator ?:

```
salary = level == "SE" ? 70000 : level == "SSE" ? 100000 : 150000;

Console.WriteLine(salary);
```

❖ If-else if-else statement

```
int salary;

string level = "SSE";

if (level == "SE")
{
    salary = 70000;
}
else if (level == "SSE")
{
    salary = 100000;
}
else
{
    salary = 150000;
}

Console.WriteLine(salary);
```

- ❖ Exception handling in Object-Oriented Programming is used to **MANAGE ERRORS**.
1. **TRY** – A try block is a block of code inside which any error can occur.
 1. **CATCH** – When any error occur in TRY block then it is passed to catch block to handle it.
 2. **FINALLY** – The finally block is used to execute a given set of statements, whether an exception occur or not.

```
try
{
    int i = 0;
    int j = 0;

    int k = i / j;
}
catch (Exception ex)
{
    //.LogError(ex.Message)
    Console.WriteLine(ex.Message);

}
finally
{
    //object.Dispose()
    Console.WriteLine("Finally");
}
//Output: Attempted to divide by zero.
//Finally
```

❖ NO

*We can write multiple catch blocks but when we will run the application and if any error occur, **only one** out of them will execute based on the type of error.*

```
try
{
    int i = 0;
    int j = 0;

    int k = i / j;
}
catch (ArithmetcException ex)
{
    Console.WriteLine("Alert");
    Console.WriteLine(ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}
```

- ❖ Finally block is mostly used to dispose the unwanted objects when they are no more required. This is good for performance, otherwise you have to wait for garbage collector to dispose them.

```
SqlConnection con = new SqlConnection("conString");
try
{
    con.Open();

    //Some logic

    // Error occurred

    con.Close();
}
catch(Exception ex)
{
    // error handled
}
finally
{
    // Connection closed
    con.Close();
}
```

- ❖ YES - We can have only try block without catch block, but then we must have **finally** block with try block.

```
.....our application...
```

```
static void Main(string[] args)
{
    SqlConnection con = new SqlConnection("conString");
    try
    {
        con.Open();
        Random rnd = new Random();
        int num = rnd.Next();

        if(num == 5)
        {
            return;
            //After this nothing will execute
        }
        //con.Close();
    }
    finally
    {
        con.Close();
    }
}
```

- ❖ Finally, is used in exception handling.
- ❖ Finalize is a method which is automatically called by the **garbage collector** to dispose the no longer needed objects.

- ❖ Throw ex will change the stack trace, where as throw will preserve the whole **stack trace**.

Error:

```
at Throw_ex_and_throw.Program.DivideZeroByZero() in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 32
  at Throw_ex_and_throw.Program.Main(String[] args) in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 15
```

```
11  static void Main(string[] args)
12  {
13      try
14      {
15          DivideZeroByZero();
16      }
17      catch (Exception ex)
18      {
19          Console.WriteLine(ex.StackTrace);
20          Console.ReadLine();
21      }
22  }
23  public static void DivideZeroByZero()
24  {
25      try
26      {
27          int i = 0, j = 0;
28          int k = i / j;
29      }
30      catch (Exception ex)
31      {
32          throw ex;
33      }
34 }
```

- ❖ Throw ex will change the stack trace, where as throw will preserve the whole stack trace.

Error:

```
at Throw_ex_and_throw.Program.DivideZeroByZero() in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 28
  at Throw_ex_and_throw.Program.Main(String[] args) in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 15
```

- ❖ Its a best practice to use *throw* as it preserve the whole **stack trace**.

```
11  static void Main(string[] args)
12  {
13      try
14      {
15          DivideZeroByZero();
16      }
17      catch (Exception ex)
18      {
19          Console.WriteLine(ex.StackTrace);
20          Console.ReadLine();
21      }
22  }
23  public static void DivideZeroByZero()
24  {
25      try
26      {
27          int i = 0, j = 0;
28          int k = i / j;
29      }
30      catch (Exception ex)
31      {
32          throw;
33      }
34 }
```

Chapter 6 : OOPS & C# - Generics & Collections

Q57. Explain **Generics** in C#? When and why to use? V Imp

Q58. What are **Collections** in C# and what are their types?

Q59. What is the difference between **Array** and **ArrayList** (atleast 2)? V Imp

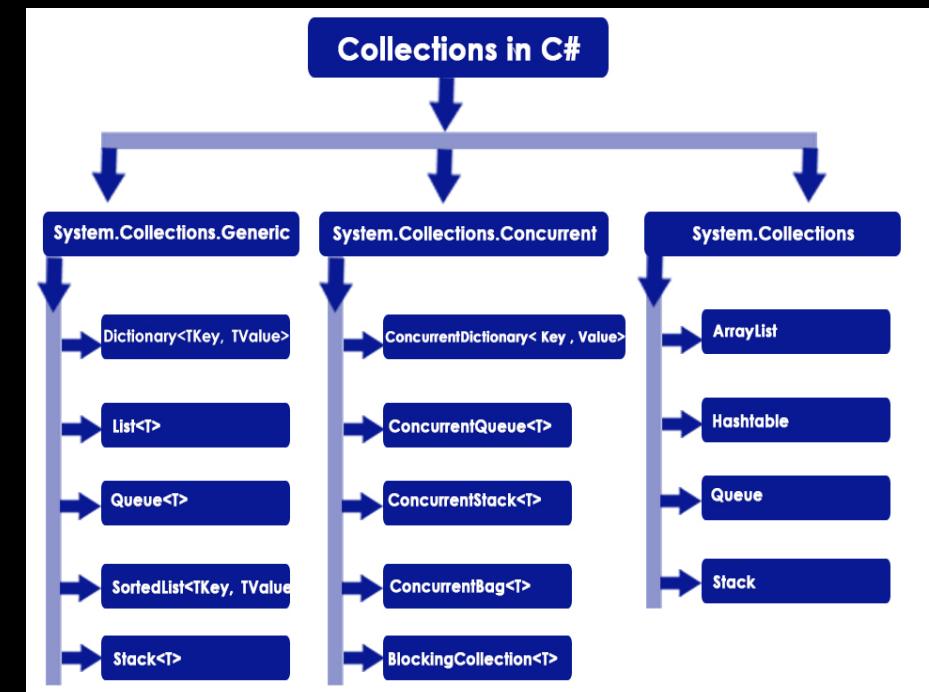
Q60. What is the difference between **ArrayList** and **Hashtable**?

Q61. What is the difference between **List** and **Dictionary** Collections? V Imp

Q62. What is **IEnumerable** in C#?

Q63. What is the difference between **IEnumerable** and **IEnumerator** in C#?

Q64. What is the difference between **IEnumerable** and **IQueryable** in C#? V Imp



- ❖ Generics allows us to make classes and methods - **type independent or type safe**.

```
public class Calculator
{
    public static bool AreEqual(int value1, int value2)
    {
        return value1.Equals(value2);
    }
}
```

```
static void Main(string[] args)
{
    bool equal = Calculator.AreEqual(4, 4);
    bool strEqual = Calculator.AreEqual("Interview", "Happy");
}
```

```
public static bool AreEqual(object value1, object value2)
{
    return value1.Equals(value2);
}
```

- ❖ The problem is, it involves Boxing from converting string (value) to object (reference) type. This will impact the performance.

- ❖ Generics allows us to make classes and methods - **type independent or type safe**.

❖ Generic Method

```
public class Calculator
{
    public static bool AreEqual<T>(T value1, T value2)
    {
        return value1.Equals(value2);
    }
}
```

```
static void Main(string[] args)
{
    //bool equal = Calculator.AreEqual(4, 4);
    //bool strEqual = Calculator.AreEqual("Interview", "Happy");

    bool equal = Calculator.AreEqual<int>(4, 4);
    bool strEqual = Calculator.AreEqual<string>("Interview", "Happy");
}
```

- ❖ Generics allows us to make classes and methods - **type independent or type safe**.

❖ Generic Class

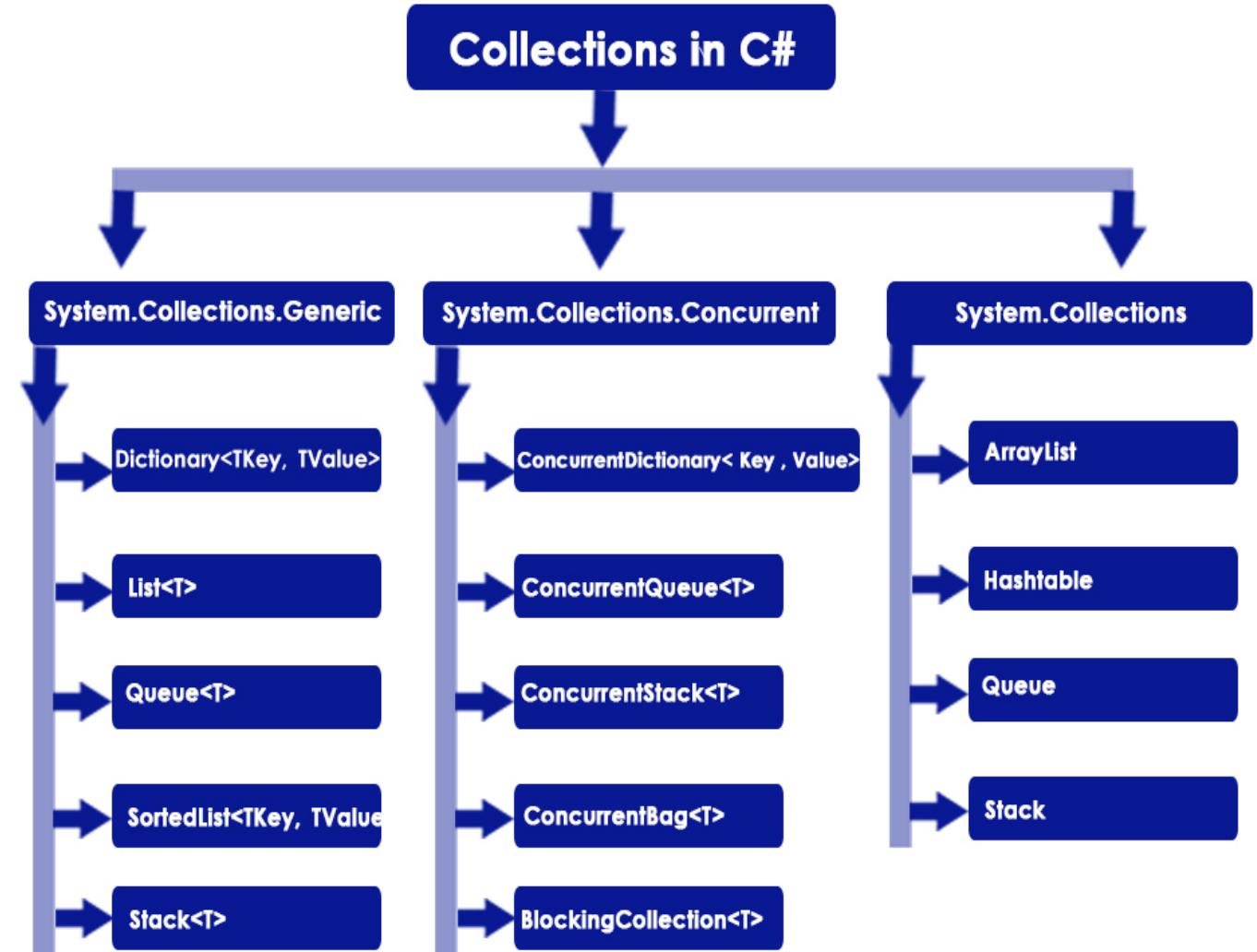
```
public class Calculator<T>
{
    public static bool AreEqual(T value1, T value2)
    {
        return value1.Equals(value2);
    }
}
```

```
static void Main(string[] args)
{
    //bool equal = Calculator.AreEqual(4, 4);
    //bool strEqual = Calculator.AreEqual("Interview", "Happy");

    bool equal = Calculator<int>..AreEqual(4, 4);

    bool strEqual = Calculator<string>..AreEqual("Interview", "Happy");
}
```

- ❖ C# collection are used to **store, manage and manipulate** data.
- ❖ For example *ArrayList*, *Dictionary*, *List*, *Hashtable* etc.



❖ Array

1. Array is **STRONGLY** typed.

This means that an array can store only specific type of items/ elements.

```
static void Main(string[] args)
{
    int[] array;

    array = new int[10];

    array[0] = 1;

    array[1] = "Happy";
}
```

2. Array can contain **FIXED** number of items.

❖ ArrayList

1. ArrayList can store **ANY** type of items\elements.

```
static void Main(string[] args)
{
    ArrayList arrayList;
    arrayList = new ArrayList();

    arrayList.Add(1);
    arrayList.Add("Happy");
}
```

2. ArrayList can store **ANY** number of items.

- ❖ In ArrayList we can only add Items/ Values to the list.

```
ArrayList arrList = new ArrayList();  
  
arrList.Add(7896);  
  
arrList.Add("Happy");
```

Value

- ❖ In Hashtable we can add Items/Values with the Keys.

```
Hashtable hashTable = new Hashtable();  
  
hashTable.Add("Number", 1);  
  
hashTable.Add("Car", "Ferrari");
```

Key Value

- ❖ List is a collection of items.
- ❖ It is the generic version of ArrayList.

```
List<string> employees = new List<string>();  
employees.Add("Happy");  
employees.Add("Rana");  
employees.Add("Roy");  
  
foreach (var employee in employees)  
{  
    Console.WriteLine(employee);  
}  
//Output: Happy Rana Roy
```

- ❖ Dictionary is a collection of key value pair.
- ❖ It is the generic version of Hashtable.

```
Dictionary<int, string> employeesD = new Dictionary<int, string>();  
employeesD.Add(123, "HappyD");  
employeesD.Add(124, "RanaD");  
employeesD.Add(125, "RoyD");  
  
foreach (KeyValuePair<int, string> emp in employeesD)  
{  
    Console.WriteLine($"{emp.Key} {emp.Value}");  
}  
  
//Output: 123 Happy 124 Rana 125 Roy
```

- ❖ **IEnumerable** interface is used when we want to **ITERATE** among our collection classes using a **FOREACH** loop.

```
static void Main(string[] args)
{
    var employees = new List<Employee>() {
        new Employee(){ Id = 1, Name="Bill" },
        new Employee(){ Id = 2, Name="Steve" }
    };

    foreach (var employee in employees)
    {
        Console.WriteLine(employee.Id + ", " + employee.Name);
    }

    Console.ReadLine();
}
```

```
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

```
namespace System.Collections.Generic
{
    ...
    public class List<T> : ICollection<T>, IEnumerable<T>, IEnumerable, IList<T>,
    {
        ...
        public List();
        ...
        public List(IEnumerable<T> collection);
        ...
        public List(int capacity);
    }
}
```

- ❖ **IEnumerable** internally uses **IEnumerator** only to iterate the collection via `foreach` loop.
- ❖ **IEnumerable** simplifies the use of **IEnumerator**.

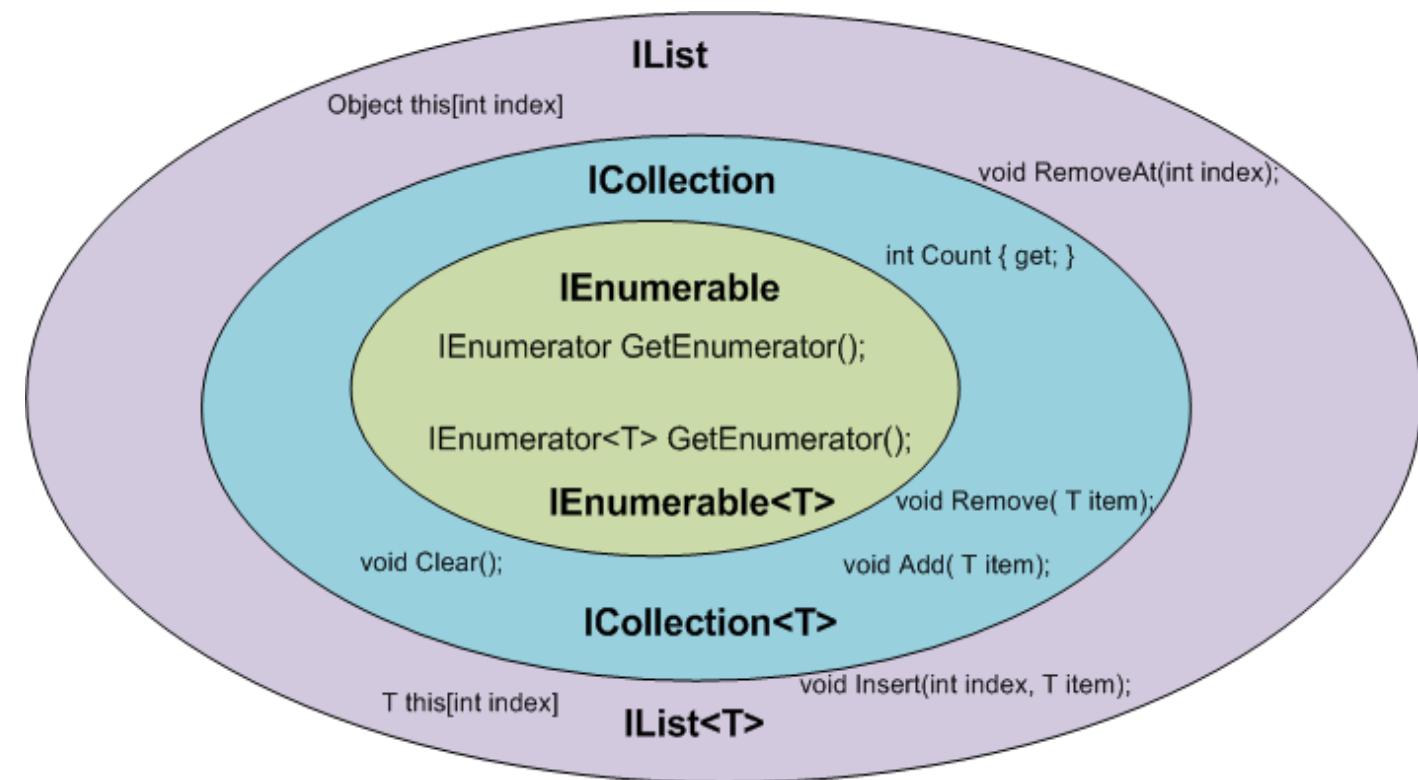
```
//IEnumerable Example
List<string> employees = new List<string>();
employees.Add("Happy");
employees.Add("Joe");
employees.Add("John");

IEnumerable<string> iEnumerableEmployees = employees;

foreach (string employee in iEnumerableEmployees)
{
    Console.WriteLine(employee);
}
//Output: Happy Joe John
```

```
//IEnumerator Example
IEnumerator<string> iEnumeratorEmployees = employees.GetEnumerator();

while (iEnumeratorEmployees.MoveNext())
{
    Console.WriteLine(iEnumeratorEmployees.Current);
}
//Output: Happy Joe John
```



```
...public interface IEnumerable
{
    //
    // Summary:
    //     Returns an enumerator that iterate
    //
    // Returns:
    //     An System.Collections.IEnumerator
    //     for the collection.
    IEnumerator GetEnumerator();
}
```

- ❖ IQueryable inherited from IEnumerable interface only, so anything you can do with a IEnumerable, you can also do with an IQueryable also.
- ❖ For example, iterating the collection can be done by both IEnumerable and IQueryable.

```
...public interface IQueryable<out T> : IEnumerable<T>, IEnumerable, IQueryable
```

```
//IEnumerable Example
List<string> employees = new List<string>();
employees.Add("Happy");
employees.Add("Joe");

IEnumerable<string> iEnumerableEmployees = employees;

foreach (string employee in iEnumerableEmployees)
{
    Console.WriteLine(employee);
}
//Output: Happy Joe John
```

```
//IQueryable Example
IQueryable<string> iQueryableEmployees = (IQueryable<string>)employees;

foreach (string employee in iQueryableEmployees)
{
    Console.WriteLine(employee);
}
//Output: Happy Joe John
```

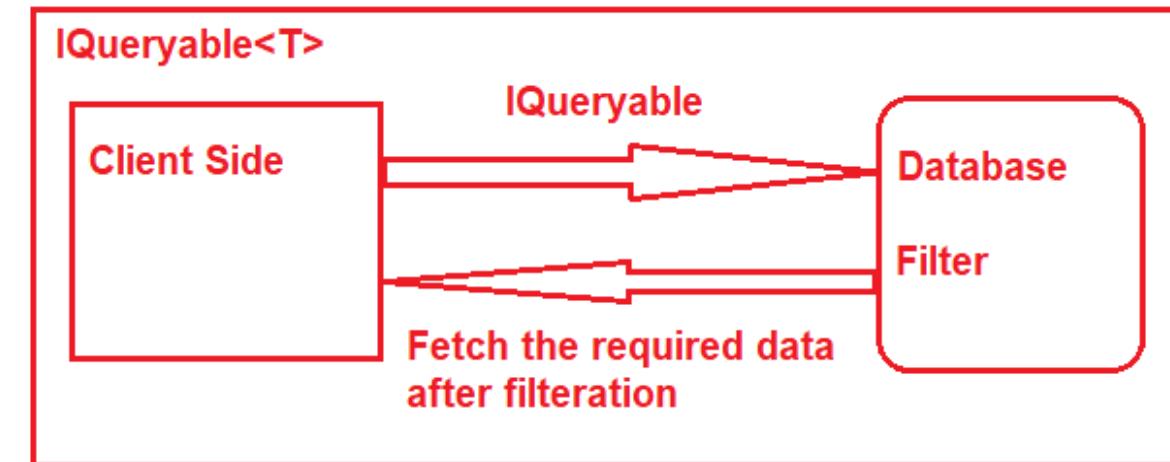
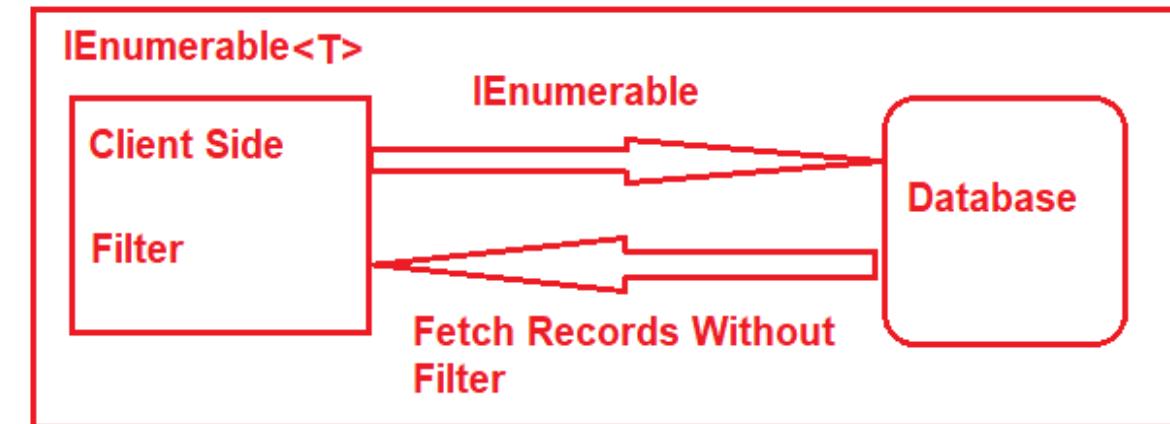
- ❖ **IEnumerable** is used with in-memory collection.
- ❖ **IQueryable** is better in getting result from database.

```
EmployeeDbContext dc = new EmployeeDbContext();

//IEnumerable Example - Better with in-memory collection
IEnumerable<Employee> listE = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

```
//IQueryable Example - Better with database interaction
IQueryable<Employee> listQ = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

- ❖ `IQueryable` inherited from `IEnumerable` interface only, so anything you can do with a `IEnumerable`, you can also do with an `IQueryable` also.
- ❖ `IEnumerable` bring all result from database and then filter it at code side, which is a network load and performance issue.
- ❖ `IQueryable` filter the result at database only and then get only filtered result, therefore less network load and better performance.
- ❖ `IQueryable` is under `SYSTEM.LINQ` namespace. `IEnumerable` is under `System.Collections` namespace.



Chapter 7 : OOPS & C# - Constructors

Q65. What is a **Constructor**? When to use constructor in real applications?

Q66. What are the types of constructor? **V Imp**

Q67. What is **Default** constructor?

Q68. What is **Parameterized** constructor?

Q69. What is **Static** constructor? What is the use in real applications?

Q70. Can we have parameters or access modifier in static constructor?

Q71. What is **Copy** constructor?

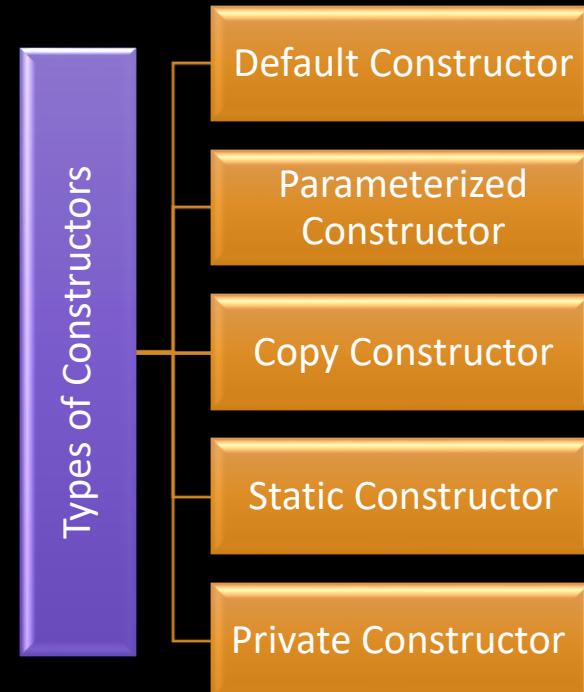
Q72. What is **Private** constructor? What is the use?

Q73. What is **Constructor** overloading?

Q74. What is **Destructor**?

Q75. Can you create object of class with private constructor in C#?

Q76. If base class and child class both have constructor which one will be called first, when derived class object is created?

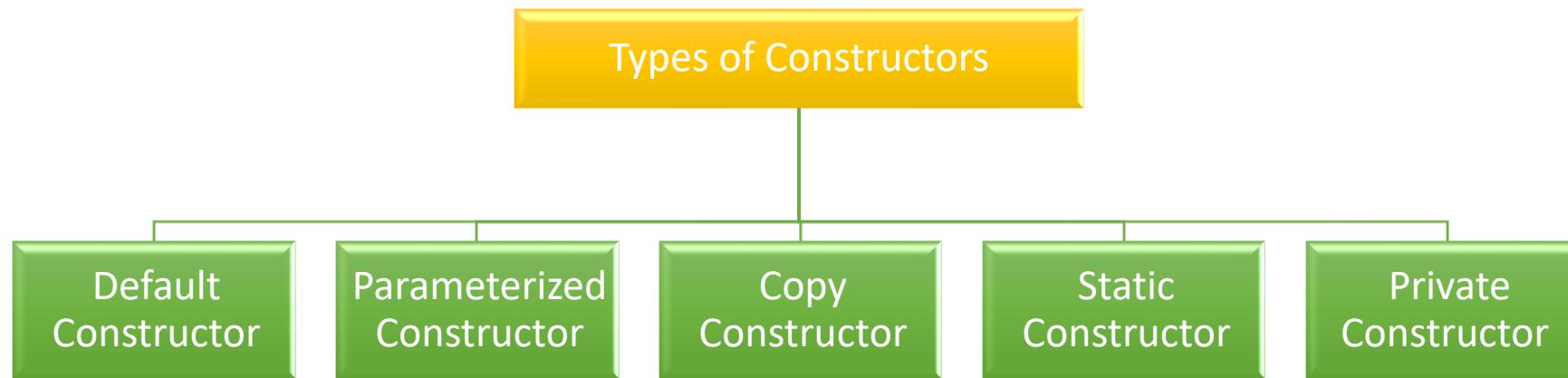


- ❖ A constructor is a **specialized method** in the class which gets executed when a class object is created.
- ❖ Constructor name will same as of Class name.
- ❖ A constructor is used **to set default values** for the class.

```
public class Employee
{
    public Employee()
    {
        Console.WriteLine("CompanyABC");
    }
}
```

```
public class Program
{
    static void Main(string[] args)
    {
        Employee emp = new Employee();

        //Output: CompanyABC
    }
}
```



- ❖ A default constructor is a constructor that is automatically created by the compiler, if no other constructors are defined in the class.

```
public class MyClass
{
    public MyClass()
    {
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        MyClass myClass = new MyClass();
    }
}
```

- ❖ A constructor with at least one parameter is a parameterized constructor.

```
public class MyClass
{
    public MyClass(int x)
    {
        Console.WriteLine(x);
    }
}
```

```
public class Program
{
    static void Main(string[] args)
    {
        MyClass myClass = new MyClass(100);
    }
}

//Output: 100
```

- ❖ Static constructor is used to be called before any static member of a class is called.

```
public class MyClass
{
    //Static constructor
    static MyClass()
    {
        Console.WriteLine("Constructor");
    }

    public static void Print()
    {
        Console.WriteLine("Method");
    }
}
```

```
static void Main(string[] args)
{
    MyClass.Print();
}

//Output
//Constructor
//Method
```

- ❖ No, static constructor can not have parameters or access modifier.

```
public class MyClass
{
    //Static constructor
    static MyClass()
    {
        Console.WriteLine("Constructor");
    }

    public static void Print()
    {
        Console.WriteLine("Method");
    }
}
```

- ❖ The constructor which creates an object by copying variables from another object is called a copy constructor.

```
static void Main(string[] args)
{
    MyClass myClass = new MyClass("Happy");

    MyClass myClassCopy = new MyClass(myClass);

    Console.WriteLine(myClassCopy.name);
}

//Output: Happy
```

```
public class MyClass
{
    public string name;

    //Parametrized Constructor
    public MyClass(string name)
    {
        this.name = name;
    }

    //Copy Constructor
    public MyClass(MyClass copy)
    {
        name = copy.name;
    }
}
```

- ❖ When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class.

```
public class MyClass
{
    private MyClass()
    {
        Console.WriteLine("Private");
    }
}
```

```
static void Main(string[] args)
{
    MyClass myClass = new MyClass();
}
```

- ❖ Constructor Overloading is a technique to define multiple constructors within a class with different sets of parameters.

```
public class Employee
{
    public Employee()
    {
        Console.WriteLine("Blank");
    }
    public Employee(int id)
    {
        Console.WriteLine(id);
    }
    public Employee(int id, string name)
    {
        Console.WriteLine(id + " " + name);
    }
}
```

```
static void Main(string[] args)
{
    Employee employee = new Employee();
    Employee employee1 = new Employee(30);
    Employee employee2 = new Employee(50, "Happy");
}
//Output: Blank
//30
//50 Happy
```

- ❖ Constructors in C# are methods inside the class used to destroy instances of that class when they are no longer needed.
- ❖ The Destructor is called implicitly by the .NET Framework's Garbage collector

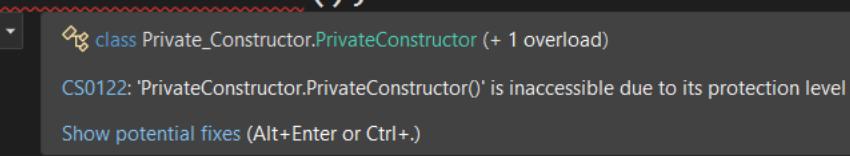
```
public class Employee
{
    public Employee()
    {
    }

    ~Employee()
    {
    }
}
```

❖ No

```
public class PrivateConstructor
{
    private PrivateConstructor()
    {
        Console.WriteLine("Private Constructor Called");
    }
}
```

```
static void Main(string[] args)
{
    PrivateConstructor privateConstructor = new PrivateConstructor();
}
```



❖ Base class

```
static void Main(string[] args)
{
    DerivedClass obj = new DerivedClass();
    Console.ReadLine();
}
//Output
//Base
//Derived
```

```
class BaseClass
{
    public BaseClass()
    {
        Console.WriteLine("Base");
    }
}

class DerivedClass : BaseClass
{
    public DerivedClass()
    {
        Console.WriteLine("Derived");
    }
}
```

Chapter 8 : OOPS & C# - Method Parameters, Delegates & Events

Q77. What is a **Method** in C#?

Q78. What is the difference between **Pass by Value** and **Pass by Reference** Parameters?

Q79. How to return more than one value from a method in C#? V Imp

Q80. What is the difference between “**out**” and “**ref**” parameters? V Imp

Q81. What is “**params**” keyword? When to use params keyword in real applications?

Q82. What are **optional parameters** in a method?

Q83. What are **named parameters** in a method?

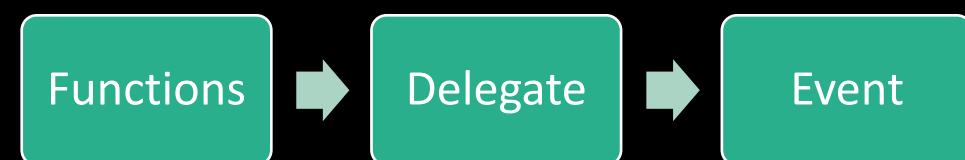
Q84. What are **Extension Methods** in C#? When to use extension methods? V Imp

Q85. What are **Delegates** in C#? When to use delegates in real applications? V Imp

Q86. What are **Multicast Delegates**?

Q87. What are **Anonymous Delegates** in C#?

Q88. What are the differences between **Events** and **Delegates**? V Imp



- ❖ A **method** is a code block that contains a series of statements.

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

    public void CalculateSalary()
    {
        int salary = Experience * 300000;

        Console.WriteLine(salary);
    }
}
```



Method

```
//Passing by Value

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x + " " + y);
    PassByValue(x, y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 5 10
}

static void PassByValue(int x, int y)
{
    x = 100;
    y = 200;
}
```

```
//Passing by Reference

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x + " " + y);
    PassByReference(ref x, ref y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 100 200
}

static void PassByReference(ref int x, ref int y)
{
    x = 100;
    y = 200;
}
```

- ❖ By using ref and out keywords.
- ❖ By using a List, ArrayList(any collection) in the return type.

```
//Passing by Reference

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x + " " + y);
    PassByReference(ref x, ref y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 100 200
}

static void PassByReference(ref int x, ref int y)
{
    x = 100;
    y = 200;
}
```

- ❖ When to use out and when to use ref?

Use out parameter to return a new and fresh value and use ref to modify an existing value.

1. No need to initialize out parameter before passing it.

1. Must initialize ref parameter else error.

2. Out parameter must be initialized before returning.

2. For Ref parameter, Initialization is not necessary before returning.

```
static void Main(string[] args)
{
    int a;
    int b = 5;

    WithRefOut obj = new WithRefOut();

    int x = obj.Update(out a, ref b);

    Console.WriteLine(x);
    Console.ReadLine();
}

public class WithRefOut
{
    public int Update(out int c, ref int d)
    {
        c = 100;
        return c + d;
    }
}
```

- ❖ Params keyword is used as a parameter which can take the **VARIABLE** number of parameters.
- ❖ It is useful when programmer don't have any prior knowledge about the number of parameters to be used.

```
static void Main(string[] args)
{
    int sum = Add(5, 10, 15, 20, 30, 40);

    Console.WriteLine(sum);
    Console.ReadLine();
}

public static int Add(params int[] numbers)
{
    int total = 0;

    foreach (int i in numbers)
    {
        total += i;
    }
    return total;
}

//Output: 120
```

You can pass any number of parameters here.

- ❖ Optional parameters allow some arguments which are not mandatory to pass, and their default value is set.

```
static void Main(string[] args)
{
    AddOptional(10, 20);

    AddOptional(10, 20, 30);
}
```

```
public static void AddOptional(int i, int j, int k = 50)
{
    Console.WriteLine(i + j + k);
}
```

- ❖ Named parameters are used to specify an argument based on the name of the argument and not the position

```
static void Main(string[] args)
{
    //Normal way
    Add(5, 10, 8, 2, 3, 6);

    //Named Parameters
    Add(a: 2.0, b: 3, c: 6.0, x: 5, y: 10, z: 8);
}
```

```
public static void Add(int x, float y, int z, double a, decimal b, double c)
{
}
```

- ❖ Extension method allows you to add new methods in the **existing class without modifying** the source code of the original class.
- ❖ Extension method must be static because this will be directly called from the class name, not by the object creation.
- ❖ **this** keyword is used for binding this method with the main class.
- ❖ USE - Use them when you want to add a method in a class which code you don't have.

```
static void Main(string[] args)
{
    string test = "HelloWorld";

    string left = test.Substring(0, 5);

    Console.WriteLine(left);

    string right = test.RightSubstring(5);

    Console.WriteLine(right);

    //Output: Hello World
}
```

```
public static class StringExtensions
{
    public static string RightSubstring(this String s, int count)
    {
        return s.Substring(s.Length - count, count);
    }
}
```

- ❖ A Delegate is a variable that holds the reference to a method or Pointer to a method.
- ❖ A delegate can refer to more than one methods of same return type and parameters.
- ❖ When to use delegate?

When we need to pass a method as a parameter.

```
delegate void Calculator(int x, int y);  
  
class Program  
{  
    public static void Add(int a, int b)  
    {  
        Console.WriteLine(a + b);  
    }  
    public static void Mul(int a, int b)  
    {  
        Console.WriteLine(a * b);  
    }  
  
    static void Main(string[] args)  
    {  
        //Instantiating Delegate  
        Calculator calc = new Calculator(Add);  
  
        //Calling method using delegate  
        calc(20, 30);  
    }  
}
```

- ❖ A Multicast Delegate in C# is a delegate that holds the references of more than one function.

```
delegate void Calculator(int x, int y);

class Program
{
    public static void Add(int a, int b)
    {
        Console.WriteLine(a + b);
    }

    public static void Mul(int a, int b)
    {
        Console.WriteLine(a * b);
    }

    static void Main(string[] args)
    {
        Calculator calc = new Calculator(Add);

        calc += Mul;

        calc(20, 30);
    }

    //Output: 50 600
}
```

- ❖ Delegates pointing methods without name are called anonymous delegates.

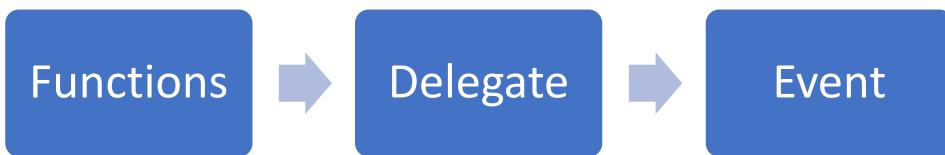
```
public delegate void Calculator(int x, int y);

class Program
{
    static void Main(string[] args)
    {
        Calculator calcAdd = delegate(int a, int b)
        {
            //Inline content of the method;
            Console.WriteLine(a + b);
        };

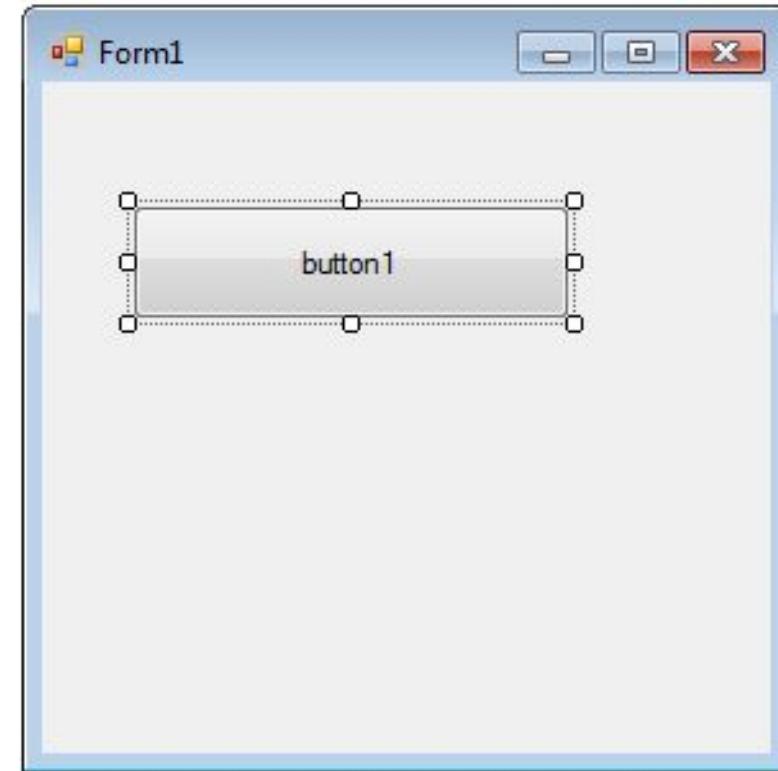
        calcAdd(20, 30);
    }
}

//Output: 50
```

- ❖ The event is a notification mechanism that depends on delegates



- ❖ An event is dependent on a delegate and cannot be created without delegates.
- ❖ Event is like a wrapper over the delegate to improve its security.



Chapter 9: OOPS & C# - Important Keywords

Q89. What is **'this'** keyword in C#? When to use it in real applications?

Q90. What is the purpose of **"using"** keyword in C#? V Imp

Q91. Can we use Using keyword with other classes apart from DBConnection?

Q92. What is the difference between **"is"** and **"as"** operators?

Q93. What is the difference between **" Readonly"** and **" Constant"** variables? V Imp

Q94. What is **"Static"** class? When to use static class in real application?

Q95. What is the difference between **"var"** and **"dynamic"** in C#?

Q96. What is **Enum** keyword used for?

Q97. Is it possible to inherit Enum in C#?

Q98. What is the use of **Yield keyword** in C#?

abstract	bool	break	byte	case	catch
char	class	const	continue	default	double
enum	else	false	finally	float	for
foreach	goto	if	int	interface	long
namespace	new	public	private	protected	return
sbyte	short	static	string	struct	switch
throw	true	try	ushort	void	while

- ❖ *this* keyword is used to refer to the CURRENT INSTANCE of the class.

```
class Program
{
    static void Main(string[] args)
    {
        Student std1 = new Student(001, "Jack");
        std1.GetStudent();
    }
}
```

- ❖ *this* keyword avoids the name confusion between class fields and constructor parameters.

```
class Student
{
    public int id;
    public string name;
    public Student(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
    public void GetStudent()
    {
        Console.WriteLine(id + " : " + name);
    }
}
```

- ❖ There are two purpose of using keyword in C#:

1. USING DIRECTIVE

```
using System;
using System.Data.SqlClient;
```

2. USING STATEMENT

- The using statement ensures that DISPOSE() method of the class object is called even if an exception occurs.

```
static void Main(string[] args)
{
    using(var connection = new SqlConnection("ConnectionString"))
    {
        var query = "UPDATE YourTable SET Property = Value";
        var command = new SqlCommand(query, connection);

        connection.Open();
        command.ExecuteNonQuery();

        //connection.Dispose();
    }
}
```

```
... public sealed class SqlConnection : DbConnection, ICloneable
{
```

```
... public abstract class DbConnection : Component, IDbConnection, IDisposable, IAsyncDisposable
{
```

[back to chapter index](#)

- ❖ Yes, using keyword can be used with any class which is inherited from **IDisposable** class. For example, with **StreamReader** class.

- ❖ The **IS** operator is **USED TO CHECK** the type of an object.
- ❖ The **AS** operator is used to **PERFORM CONVERSION** between compatible reference type.

```
static void Main(string[] args)
{
    int i = 5;

    bool check = i is int;

    Console.WriteLine(check);

    //Output: true
}
```

```
static void Main(string[] args)
{
    object obj = "Hello";

    string str1 = obj as string;

    Console.WriteLine(str1);

    //Output: Hello
}
```

- ❖ The **IS** operator will return boolean type.
- ❖ The **AS** operator is not of boolean type.

❖ Constant

```
class Example
{
    public const int myConst = 10;

    public const int myConst1;

    public Example(int b)
    {
        myConst1 = 20;
    }
}
```

❖ Readonly

```
class Example
{
    public readonly int myReadonly1 = 100;

    public readonly int myReadonly2;

    public Example(int b)
    {
        myReadonly2 = b * 100;
    }
}
```

1. Using constant, we must assign values with declaration itself. But readonly fields can be assigned in declaration as well as in the constructor part.
2. Constant field value cannot be changed, but Readonly field value can be changed.
3. “const” keyword for Constant and “readonly” keyword is used for Readonly.
4. Constant is a COMPILE time constant, and ReadOnly is a RUNTIME constant.

- ❖ A static class is a class which object can not be created, and which can not be inherited.
- ❖ Use of static class:

Static classes are used as **containers** for static members like methods, constructors and others.

```
public static class MyCollege
{
    //static fields
    public static string collegeName;
    public static string address;

    //static constructor
    static MyCollege()
    {
        collegeName = "ABC College";
    }

    // static method
    public static void CollegeBranch()
    {
        Console.WriteLine("Computers");
    }
}
```

- ❖ VAR - The type of the variable is decided by the compiler at **compile time**.
- ❖ DYNAMIC - The type of the variable is decided at **run time**.

```
static void Main(string[] args)
{
    var a = 10;

    a = "Interview";

    Console.WriteLine(a);
}
```

```
static void Main(string[] args)
{
    dynamic b = 10;

    b = "Happy";

    Console.WriteLine(b);
}

//Output: Happy
```

- ❖ An **enum** is a special "class" that represents a group of constants.

```
class LevelConstants
{
    public const int Low = 0;

    public const int Medium = 1;

    public const int High = 2;
}
```

```
enum Level
{
    Low,
    Medium,
    High
}
```

```
enum Weekdays
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}
```

```
static void Main(string[] args)
{
    Level myLevel = Level.Medium;

    Console.WriteLine(myLevel);
}
```

- ❖ Enum is a **sealed** class type, so it cannot be inherited.

```
enum Level
{
    Low,
    Medium,
    High
}

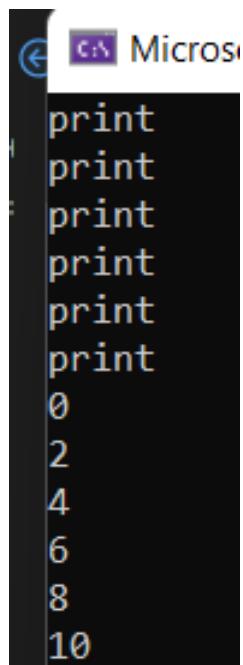
class Inherit : Level
{
    enum What_is_Enum_keyword_used_for.Level
    {
    }
}
```

CS0509: 'Inherit': cannot derive from sealed type 'Level'

```
//Input: 10  
  
//Output: 0 2 4 6 8
```

```
static IEnumerable<int> GetEvenNumbers(int upto)  
{  
    List<int> numbers = new List<int>();  
  
    for (int i = 0; i <= upto; i += 2)  
    {  
        numbers.Add(i);  
        Console.WriteLine("print");  
    }  
    return numbers;  
}
```

```
static void Main(string[] args)  
{  
    IEnumerable<int> getEventNumbers = GetEvenNumbers(10);  
  
    foreach(int eventNumber in getEventNumbers)  
    {  
        Console.WriteLine(eventNumber);  
    }  
}
```



- ❖ The yield keyword will act as an iterator blocker and generate or return values.

```
//Input: 10  
  
//Output: 0 2 4 6 8
```

```
print  
print  
print  
print  
print  
print  
print  
0  
2  
4  
6  
8  
10
```

```
0  
print  
2  
print  
4  
print  
6  
print  
8  
print  
10  
print
```

```
static IEnumerable<int> GetEvenNumbers(int upto)  
{  
    //List<int> numbers = new List<int>();  
  
    for (int i = 0; i <= upto; i += 2)  
    {  
        //numbers.Add(i);  
        yield return i;  
        Console.WriteLine("print");  
    }  
    //return numbers;  
}
```

```
static void Main(string[] args)  
{  
    IEnumerable<int> getEventNumbers = GetEvenNumbers(10);  
  
    foreach(int eventNumber in getEventNumbers)  
    {  
        Console.WriteLine(eventNumber);  
    }  
}
```

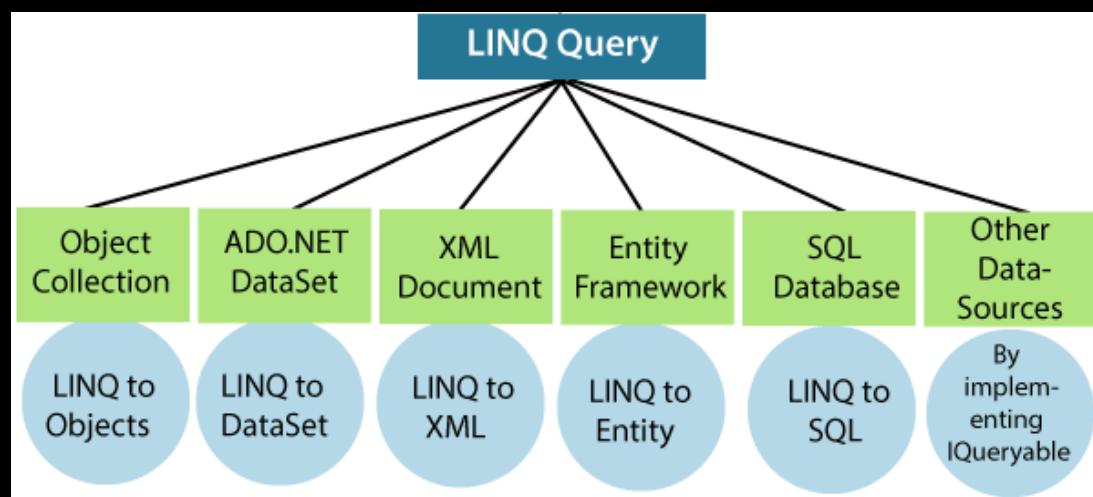
Chapter 10 : OOPS & C# - LINQ

Q99. What is **LINQ**? When to use LINQ in real applications?

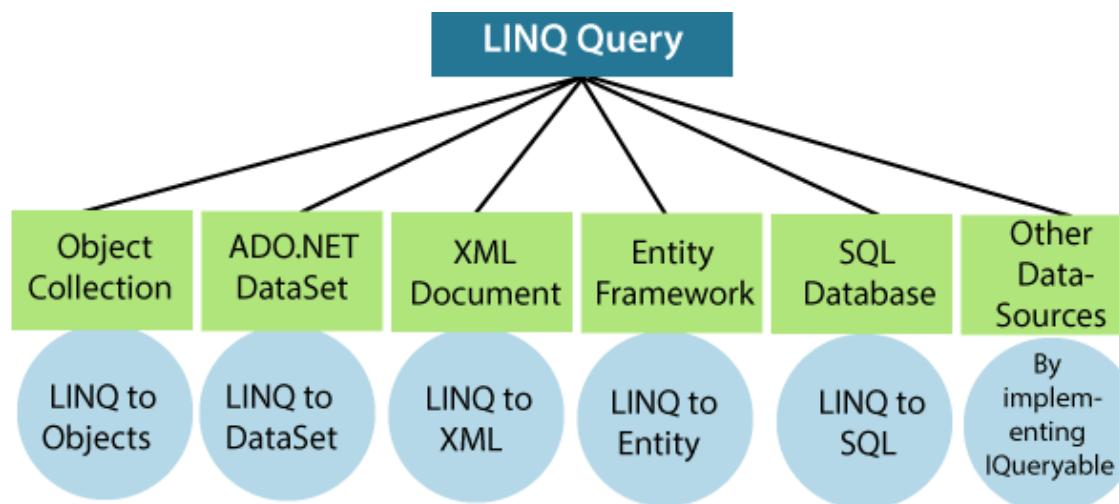
Q100. What are the advantages & disadvantages of **LINQ**?

Q101. What is **Lambda Expressions**? What is the use in real applications?

Q102. What is the difference between **First** and **FirstOrDefault** methods in **LINQ**? V Imp



- ❖ LINQ (Language Integrated Query) is uniform query syntax in C# to retrieve data from different sources.



```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

List<int> filteredNumbers = new List<int>();

foreach (int n in numbers)
{
    if (n > 2)
    {
        filteredNumbers.Add(n);
    }
}
```

```
//using System.Linq
IQueryable<int> filteredNumbers = from n in numbers
where n > 2
select n;
```

Advantages of LINQ

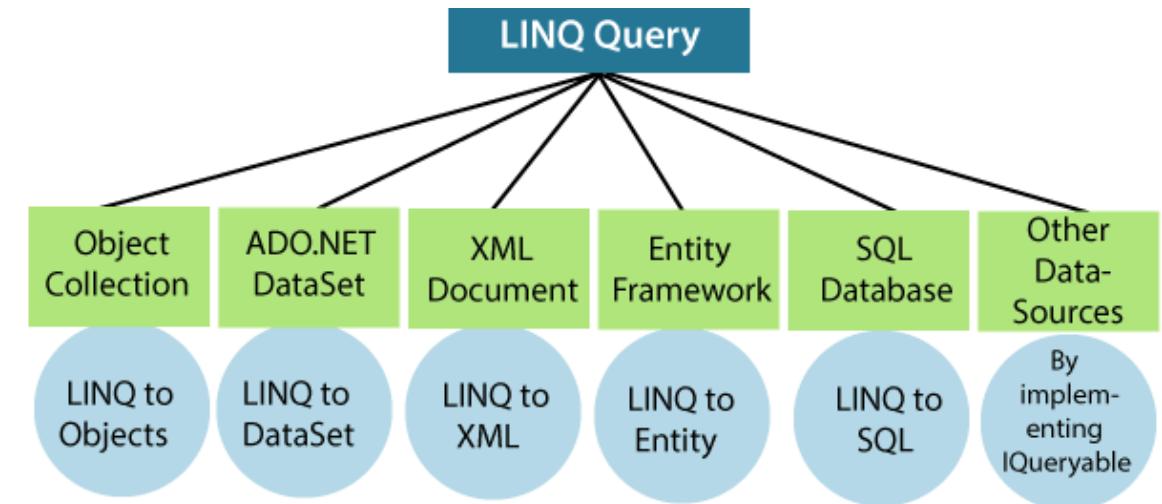
1. Easy and simple syntax to Learn
2. Improved code readability
3. Improved performance
4. Type safety

```
//using System.Linq  
IQueryable<int> filteredNumbers = from n in numbers  
where n > 2  
select n;
```

```
//using System.Linq  
IQueryable<string> filteredNumbers = from n in numbers  
where n > 2  
select n;
```

Disadvantages of LINQ

1. Limited support for some data sources
2. Difficult to maintain and debug



- ❖ A lambda expression is used to **simplify** the syntax of anonymous methods.

```
static void Main(string[] args)
{
    List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

    List<int> evenNumbers = GetEvenNumbers(numbers);
}

static List<int> GetEvenNumbers(List<int> numbers)
{
    List<int> evenNumbers = new List<int>();

    foreach (int n in numbers)
    {
        if (n % 2 == 0)
        {
            evenNumbers.Add(n);
        }
    }
    return evenNumbers;
}
```

```
//List method and Lambda expression
```

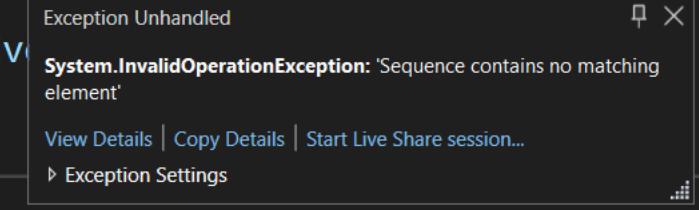
```
List<int> evenNumbers = numbers.FindAll(x => x % 2 == 0);
```

- ❖ First method will return the first value, but it is not able to handle null values.
- ❖ FirstOrDefault will return the first value, and it is able to handle null values also.

```
static void Main(string[] args)
{
    List<int> numbers = new List<int> { 1, 3, 5 };

    int firstEvenNumber = numbers.First(x => x % 2 == 0); ✖

    Console.WriteLine(firstEvenNumber);
}
```



```
static void Main(string[] args)
{
    List<int> numbers = new List<int> { 1, 3, 5 };

    int firstEvenNumber = numbers.FirstOrDefault(x => x % 2 == 0);

    Console.WriteLine(firstEvenNumber);

    //Output: 0
}
```

Chapter 11 : .NET Framework - Basics

Q103. What are the important components of .NET framework? V Imp

Q104. What is an Assembly? What are the different types of assembly?

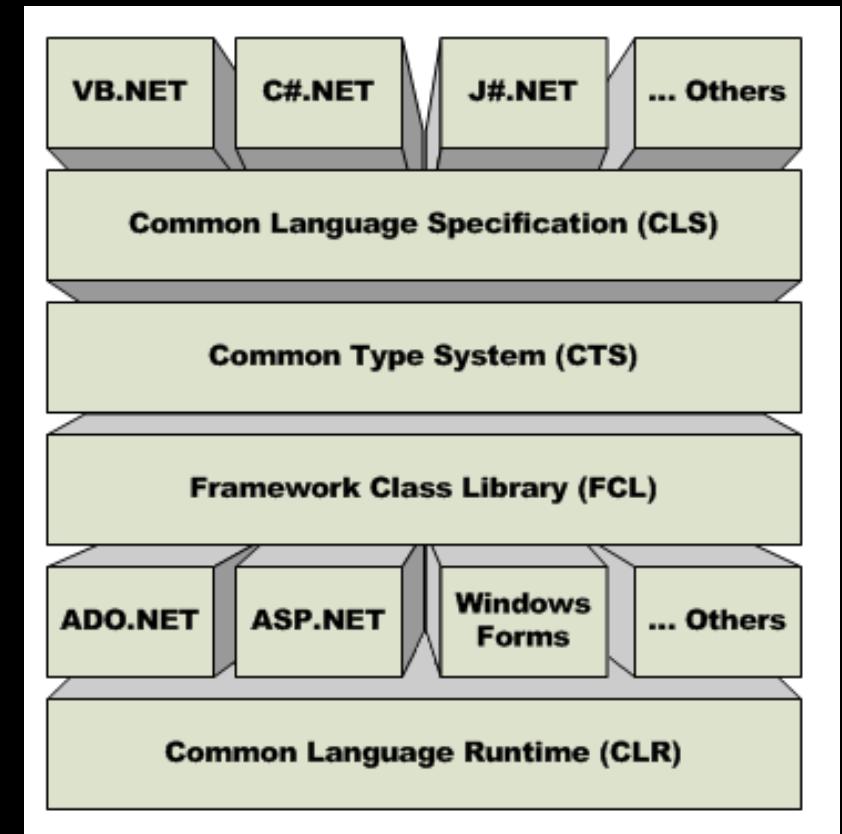
Q105. What is GAC?

Q106. What is Reflection?

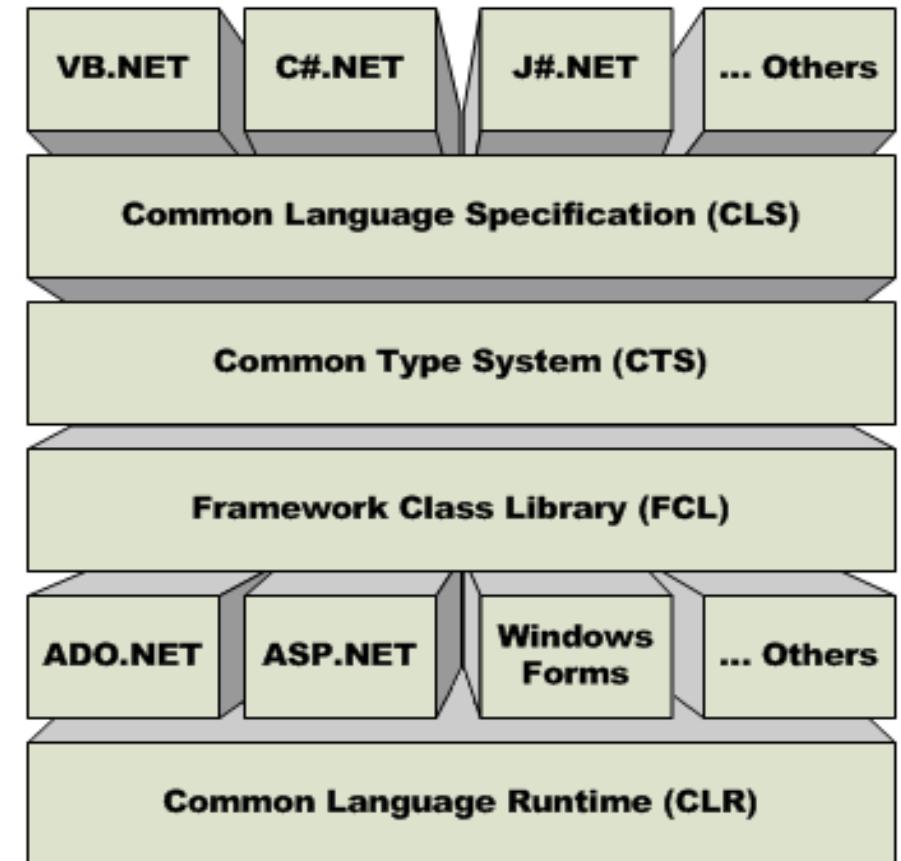
Q107. What are Serialization and Deserialization? V Imp

Q108. What is meant by Globalization and Localization?

Q109. What are Window Services?



- ❖ CLR (Common Language Runtime)
 - ❖ CLR **manages** the execution of programs for example operations like memory management, security checks etc.
 - ❖ CLR provides execution of **managed code** only (Code written in any .NET-supported language such as C#, Visual Basic .NET, or F#).
- ❖ CTS(Common Type System)
 - ❖ CTS is a set of rules that define how **types** are declared, used, and managed in the .NET Framework. Types like int, string, double.
- ❖ CLS(Common Language Specification)
 - ❖ CLS is a **subset** of CTS. It defines a set of **rules** that every language must follow which runs under .NET framework.
 - ❖ The benefit is, if you create same program logic in different languages of .NET, then the compiler will generate same dll.
- ❖ FCL(Framework Class Library)
 - ❖ FCL is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.
 - ❖ For example, String, ArrayList, List classes are provided by FCL only.

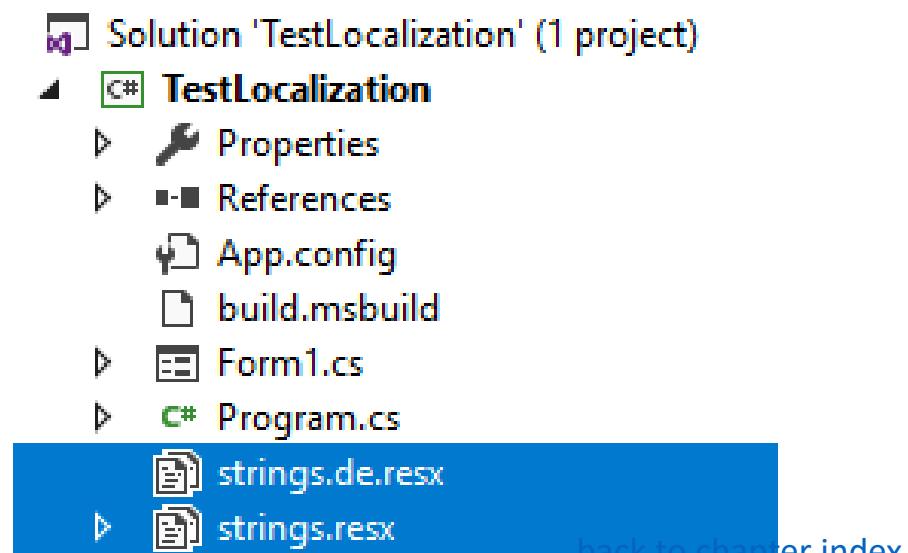
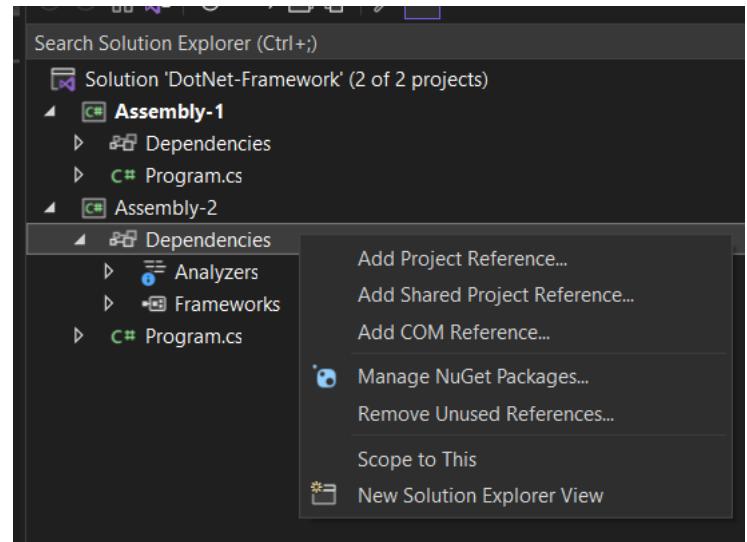


- ❖ **Assembly** is unit of deployment like EXE or a DLL.

When you create a code and build the solution, then the .NET Framework convert your code into Intermediate Language and that is placed inside the assembly(dll), which you can find inside bin folder.

- ❖ There are 3 types of assemblies:

1. **Private assembly** - A private assembly can be used by a single application only. It is not accessible outside. So, all the projects you create will by default create private assembly only.
2. **Public/ shared assembly** - Shared assemblies are usually libraries of code, which **multiple** applications can use. It is registered in the **global assembly cache(GAC)**.
3. **Satellite assembly** - A satellite Assembly is defined as an assembly with resources only, no executable code.



- ❖ GAC stands for **Global Assembly Cache**.
- ❖ GAC is the place where **public assemblies** are stored.
- ❖ Private assembly can be converted into public assembly by adding them in GAC using **gacutil** tool.

- ❖ Reflection is the ability of a code to access the metadata of the assembly during runtime.
- ❖ Metadata is information about data.



- ❖ Suppose you have to check the version of the assembly at runtime, then you can use reflection.

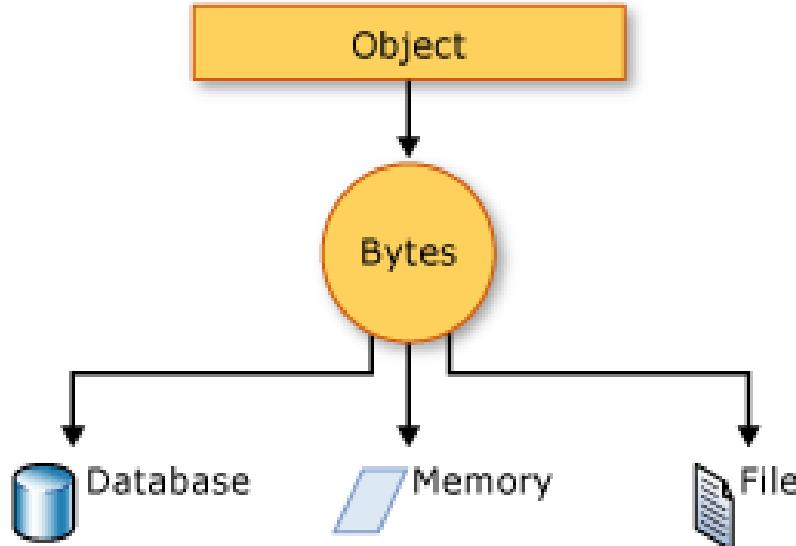
```
// Get the version number of the current assembly
Assembly assembly = Assembly.GetExecutingAssembly();
AssemblyName assemblyName = assembly.GetName();
Version version = assemblyName.Version;
string versionString = version.ToString();

// Print the version number to the console
Console.WriteLine("Assembly version: " + versionString);

Console.Read();
```

```
// Get a reference to the method using reflection
Type t = obj.GetType();
MethodInfo method = t.GetMethod(methodName);
```

- ❖ Serialization is the process of **converting an object** into a format that can be stored, transmitted, or reconstructed later.



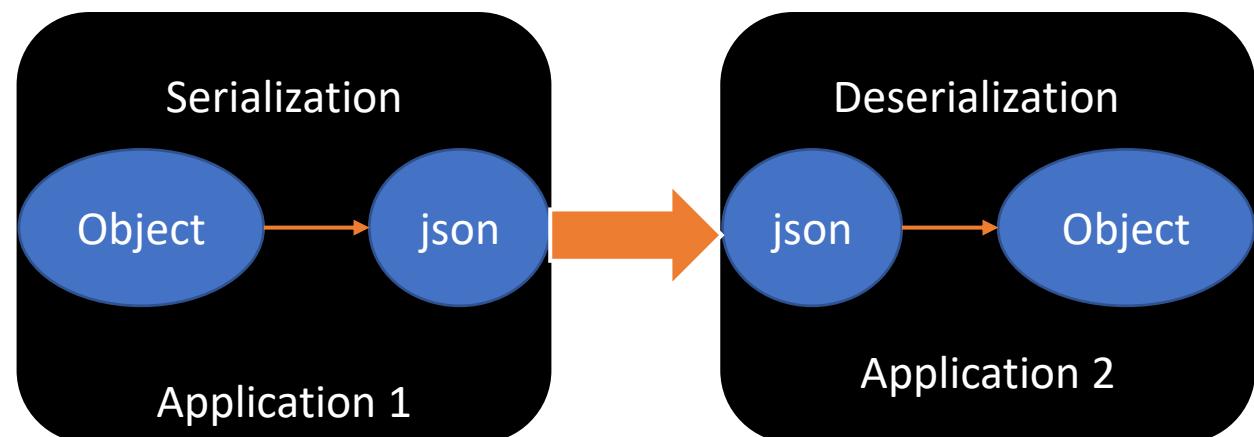
Types of Serialization

Binary
Serialization

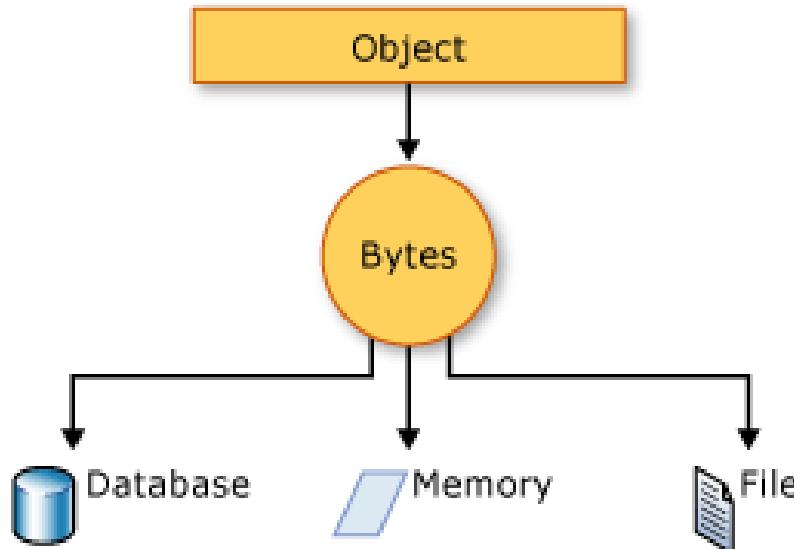
XML
Serialization

JSON
Serialization

- ❖ Deserialization is the process of converting serialized data, such as binary/ XML/ json data, back into an object.



- ❖ Serialization is the process of converting an object into a format that can be stored, transmitted, or reconstructed later.



- ❖ When to use serialization??

It is mostly used in Web API to convert class objects into JSON string.

```
Employee employee = new Employee();  
  
employee.Id = 100;  
employee.Name = "Happy";  
  
//Convert object to json  
string json = JsonConvert.SerializeObject(employee);  
  
Console.WriteLine(json);  
Console.ReadLine();  
  
//Output: {"Id":100,"Name":"Happy"}
```

- ❖ Deserialization is the process of converting serialized data, such as binary/ XML/ json data, back into an object.

- ❖ Globalization is the process of designing and developing a software product that can support different cultures and regions of the world.

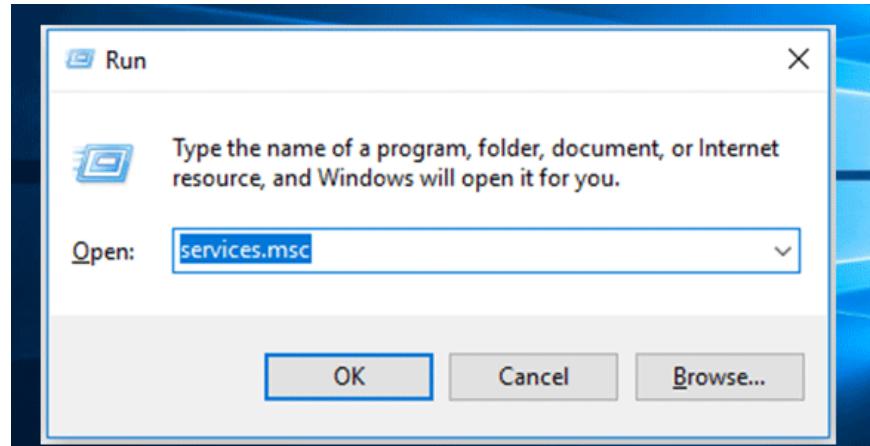


- ❖ Localization is the process of adapting a globalized application, to a particular culture/locale.

- English
- French
- German
- Spanish
- Italian
- Portuguese

-->

- ❖ Windows service is a computer program that runs in the **BACKGROUND** to execute some tasks.

A screenshot of the Windows Services (Local) console. The title bar says 'Services' and the sub-title bar says 'Services (Local)'. The main area is a table with the following columns: Name, Description, Status, Startup Type, and Log On As. The table lists numerous services, each with a small gear icon. The 'Status' column shows various states like 'Running', 'Automatic', 'Manual', and 'Disabled'. The 'Log On As' column shows 'Local System' or 'Local Service' for most services. The table has a header row and many data rows, with the last few rows partially cut off.

- ❖ Windows services can be started **AUTOMATICALLY** or manually.
- ❖ You can also manually pause, stop and restart Windows services.

Chapter 12 : .NET Framework - Garbage Collection

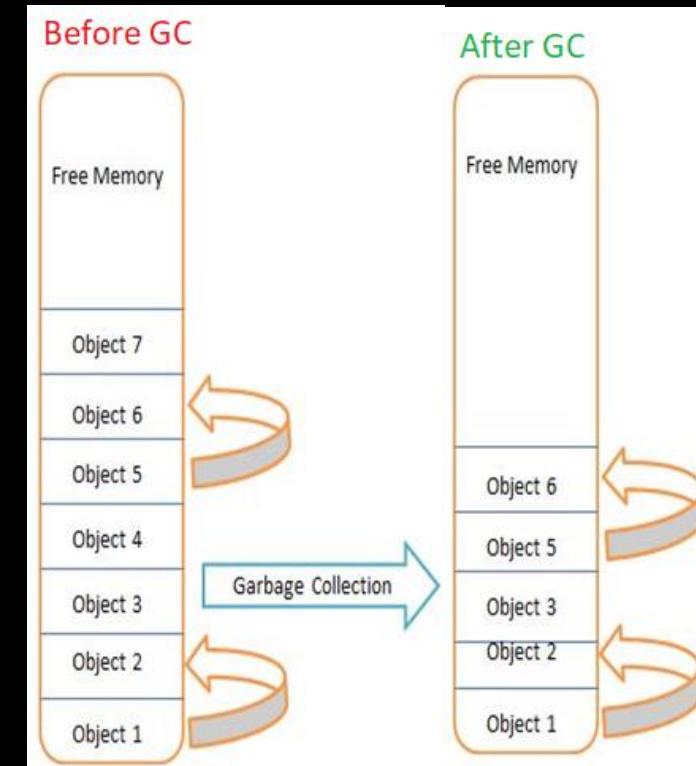
Q110. What is Garbage Collection(GC)? V Imp

Q111. What are Generations in garbage collection?

Q112. What is the difference between “Dispose” and “Finalize”? V Imp

Q113. What is the difference between “Finalize” and “Finally” methods?

Q114. Can we force Garbage Collector to run?

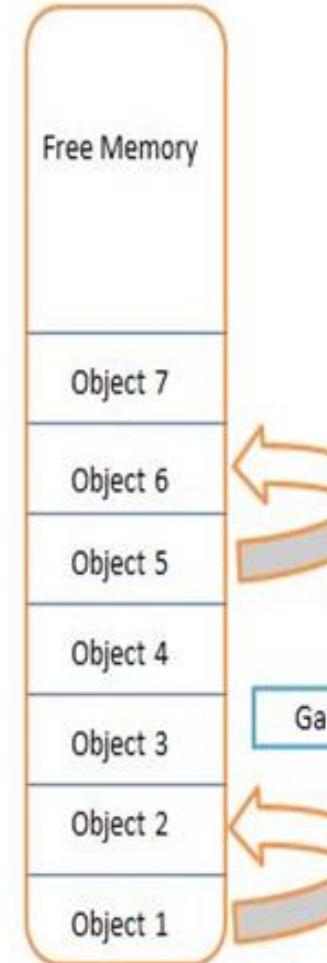


- ❖ The garbage collector (GC) **manages** the allocation and release of memory in .NET framework.
- ❖ Garbage collection is one of the responsibilities of CLR only.

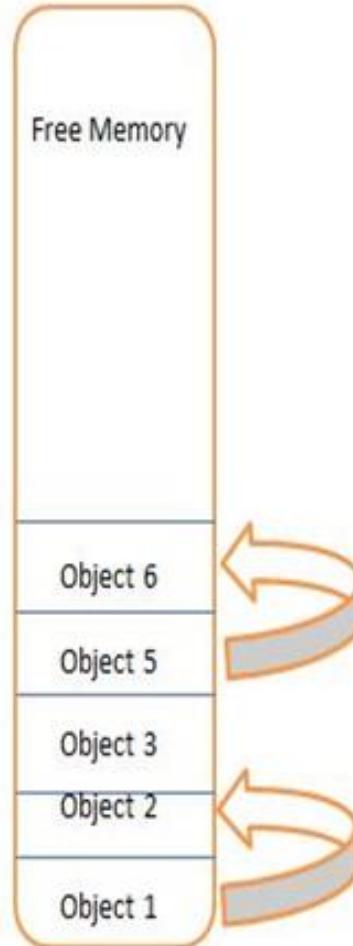
```
static void Main(string[] args)
{
    Employee employee = new Employee();
    employee.GetSalary();
}
```

- ❖ *Garbage collector will dispose this employee object automatically when it is no longer needed.*

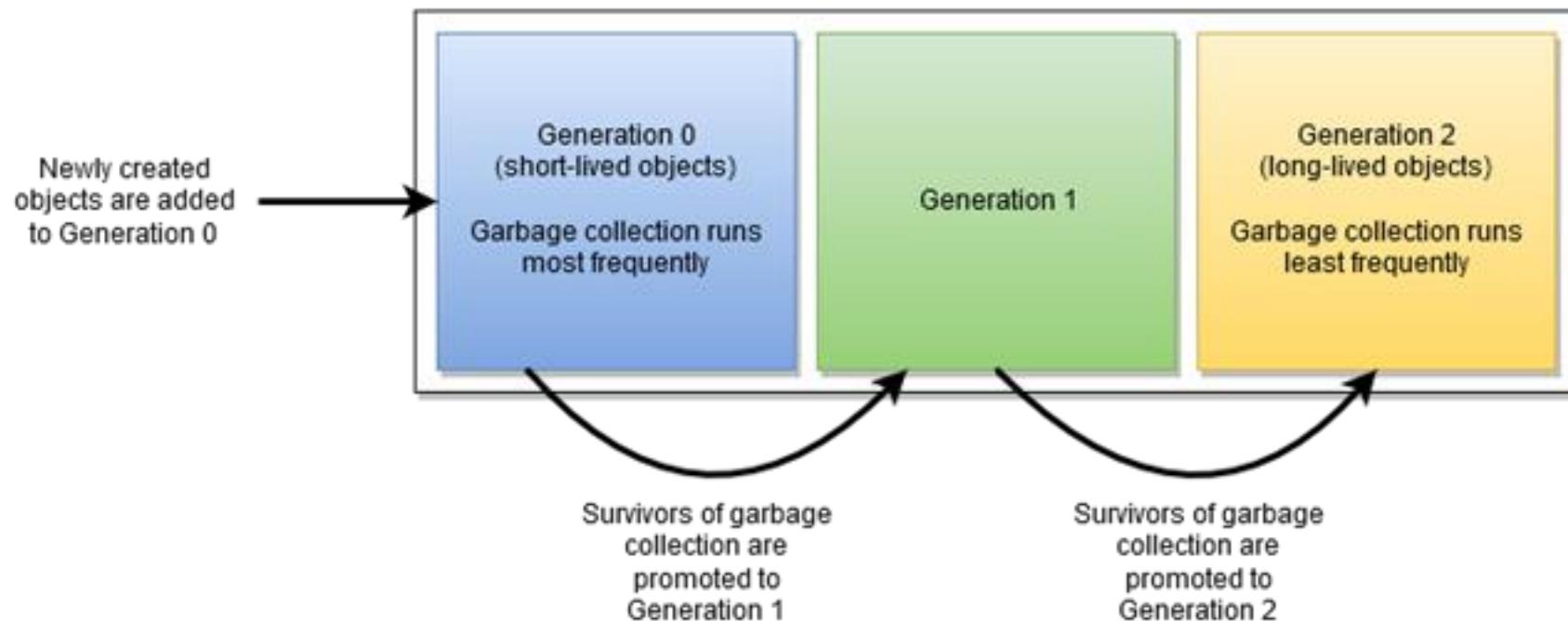
Before GC



After GC



- ❖ Garbage collection release the objects which are no longer needed.
- ❖ .NET framework has a mechanism to decide which objects are no longer needed and that mechanism we call generations.
- ❖ Generation is a mechanism to collect the short-lived objects more frequently than the longer-lived object.



- ❖ Both Dispose and Finalize methods are used to release the unmanaged objects which are no longer required.
- ❖ Finalize method is called automatically by the garbage collector.
- ❖ But the Dispose method is called explicitly by the code to release any unmanaged object.

```
using (SqlConnection connection = new SqlConnection(conString))
{
    // use the connection here
    connection.Open();

    // execute a query, etc.

} // the connection is automatically closed here
```

```
public class Employee : IDisposable
{
    private bool disposed = false;

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!disposed)
        {
            if (disposing)
            {
                // Release managed resources
            }

            // Release unmanaged resources
            disposed = true;
        }
    }
}
```

- ❖ Finalize method is used for **garbage collection**. So before destroying an object this method is called as part of clean up activity by GC.
- ❖ Finally block is used in **exception handling** for executing the code irrespective of exception occurred or not.

```
try
{
    SqlConnection con = new SqlConnection(connectionString);
    //Some logic
    //Error occurred
}
catch(ExceptionName ex) {
    //Error handled
}
finally
{
    //Connection closed
    con.Close();
}
```

- ❖ Yes, by calling **GC.COLLECT()** method we can force garbage collector to run, but this is not recommended, instead use Dispose method.

```
static void Main(string[] args)
{
    // Allocate some memory
    var obj1 = new object();
    var obj2 = new object();

    // Set obj1 and obj2 to null to make
    // them eligible for garbage collection
    obj1 = null;
    obj2 = null;

    // Trigger a garbage collection cycle
    System.GC.Collect();

    // The memory used by obj1 and obj2
    // should now be freed
}
```

Chapter 13 : .NET Framework - Threading

Q115. What is the difference between **Process** and **Thread**?

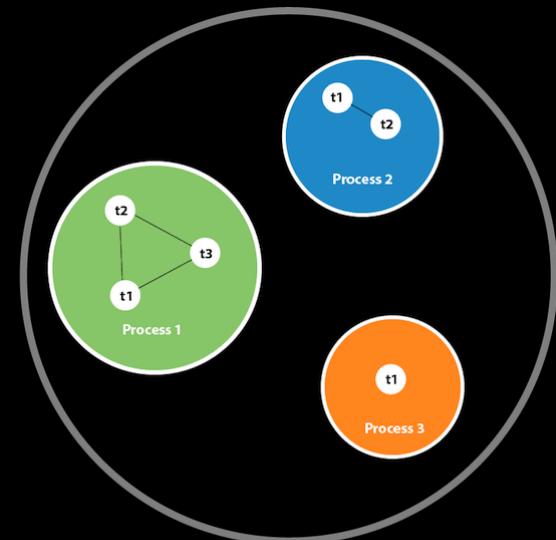
Q116. Explain **Multithreading**? V Imp

Q117. What is the difference between **synchronous** and **asynchronous** programming?

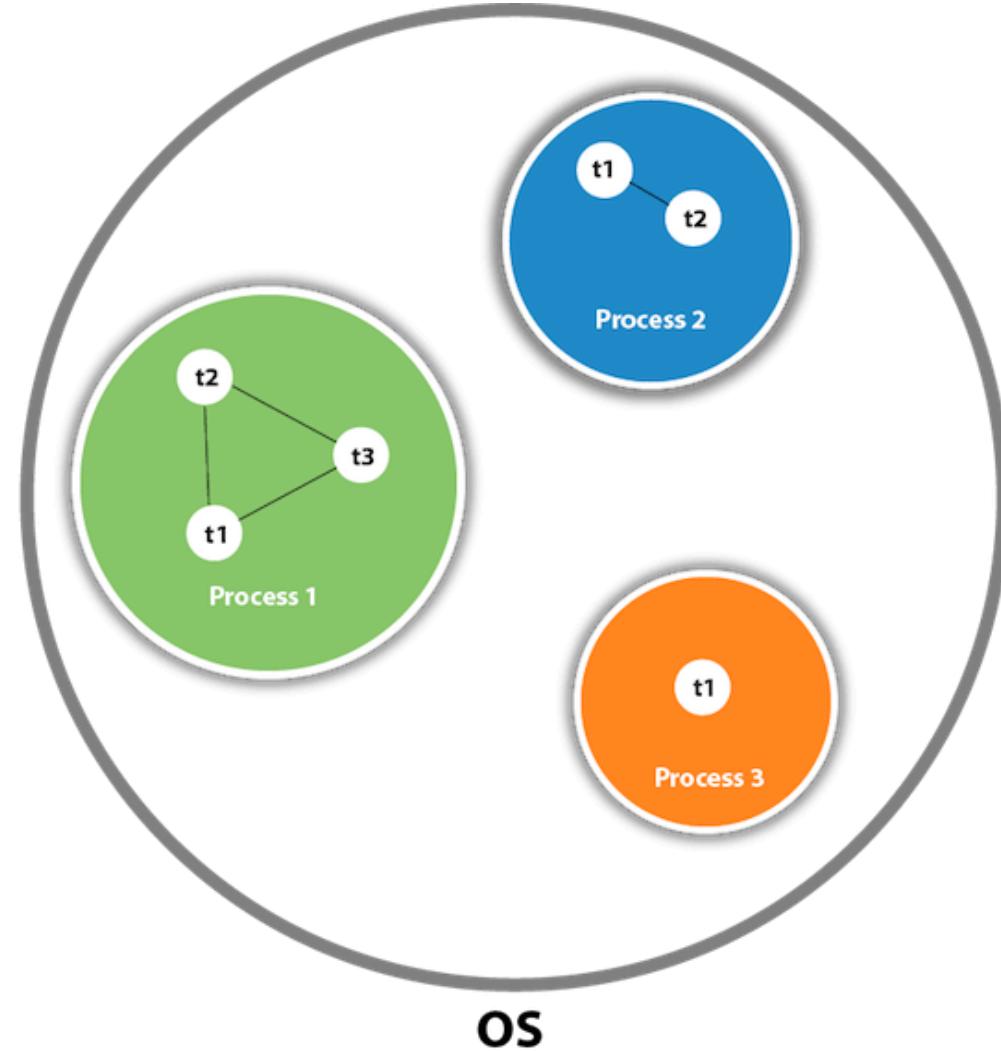
What is the role of Task? V Imp

Q118. What is the difference between **Threads** and **Tasks**? What are the advantages of Tasks over Threads?

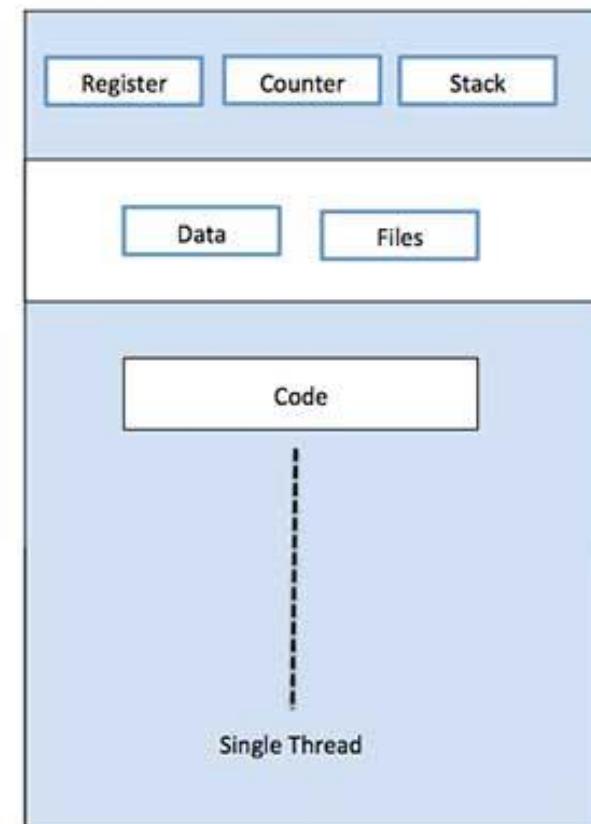
Q119. What is the role of **Async** and **Await**? V Imp



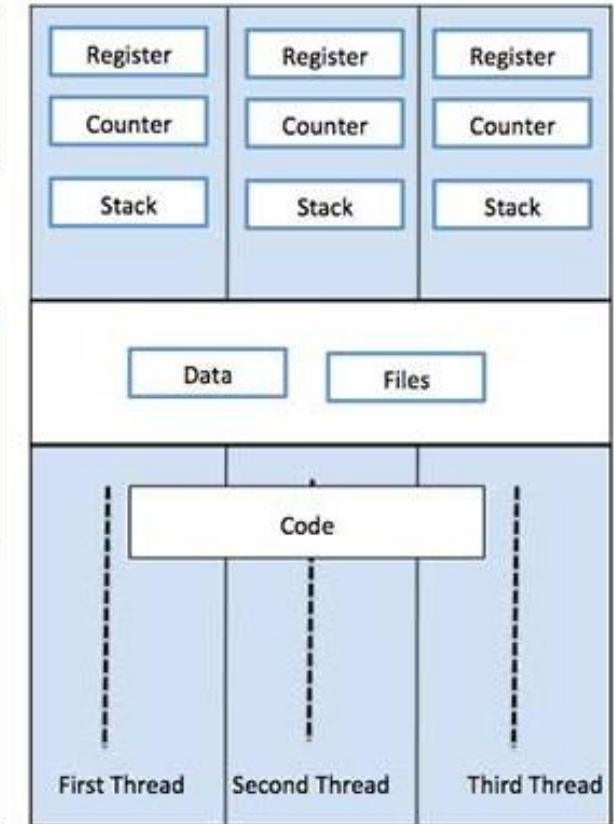
- ❖ A process is an instance of a program with its **own** memory space and system resources.
- ❖ A thread is the smallest unit of process, that **shares** memory and resources with other threads within the same process.



- ❖ Multithreading refers to the ability to execute multiple threads of code **concurrently** within a single process.
- ❖ Multithreading allows you to perform multiple tasks simultaneously, such as downloading data while displaying a progress bar.
- ❖ To create multithreaded application in C#, we need to use **SYSTEM.THREADING** namespace.



Single Process P with single thread



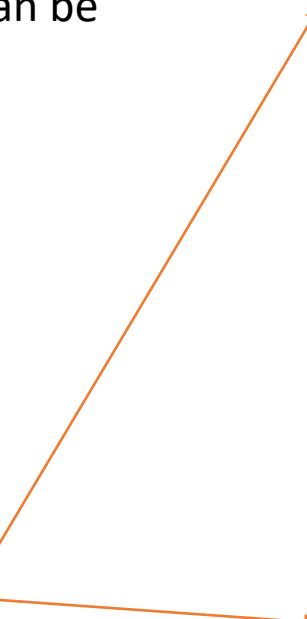
Single Process P with three threads

- ❖ In synchronous programming, task is executed in a sequential manner.
- ❖ In asynchronous programming, tasks can be executed independently of each other.

```
public static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

public static int Method2()
{
    return 20;
}

public static int Method3()
{
    return 30;
}
```



```
public static void Main(string[] args)
{
    Console.WriteLine(Method1());
    Console.WriteLine(Method2());
    Console.WriteLine(Method3());
}
//Output 10 20 30
```

Synchronous
Programming


```
public static void Main(string[] args)
{
    Task task1 = Task.Run(() => {
        Console.WriteLine(Method1());
    });

    Task task2 = Task.Run(() => {
        Console.WriteLine(Method2());
    });

    Task task3 = Task.Run(() => {
        Console.WriteLine(Method3());
    });
    Console.Read();
}
//Output: 20 30 10
```

Asynchronous
Programming

- ❖ In .NET, threads and tasks are two different ways for doing **multithreading**.
- ❖ Thread is a general programming concept. On the other hand, Microsoft created Task in .NET to **simplify** the use of Threads.
- ❖ Tasks are like a **wrapper** over Threads.
- ❖ Tasks **internally** uses threads only.



```
public class ExampleThread
{
    public void DoWork()
    {
        Thread thread = new Thread(new
            ThreadStart(LongRunningMethod));
        thread.Start();
    }

    private void LongRunningMethod()
    {
        // simulate a long-running operation
        Thread.Sleep(5000);
        // continue with the rest of the method
    }
}
```

```
public class ExampleTask
{
    public async Task DoWorkAsync()
    {
        await Task.Delay(5000);
        // continue with the rest of the method
    }
}
```

- ❖ In .NET, threads and tasks are two different ways for doing **multithreading**.
 - ❖ Thread is a general programming concept. On the other hand, Microsoft created Task in .NET to **simplify** the used of Threads.
 - ❖ Tasks are like a **wrapper** over Threads.
 - ❖ Tasks **internally** uses threads only.
- ❖ Advantages of Tasks over Threads:
 1. Simplified Code.
 2. Exception handling.
 3. A Task can **return a result**, but there is no proper way to return a result from Thread.
 4. We can apply **chaining** and **parent/ child** on multiple tasks, but it can be very difficult in threads.

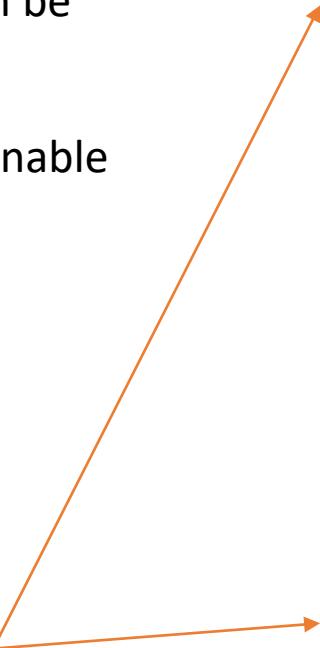


- ❖ In synchronous programming, task is executed in a sequential manner.
- ❖ In asynchronous programming, tasks can be executed independently of each other.
- ❖ Async and Await are keywords used to enable asynchronous programming.

```
public static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

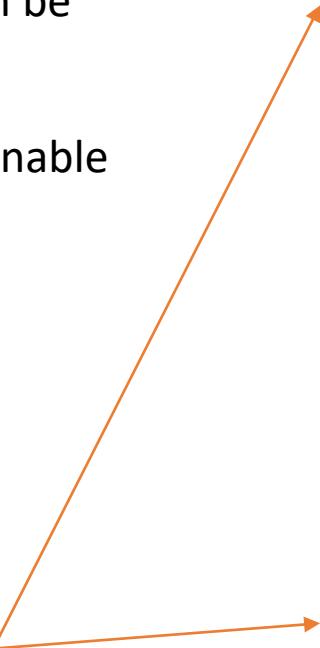
public static int Method2()
{
    return 20;
}

public static int Method3()
{
    return 30;
}
```



```
public static void Main(string[] args)
{
    Console.WriteLine(Method1());
    Console.WriteLine(Method2());
    Console.WriteLine(Method3());
}
//Output 10 20 30
```

Synchronous
Programming



```
public static void Main(string[] args)
{
    Task task1 = Task.Run(() => {
        Console.WriteLine(Method1());
    });

    Task task2 = Task.Run(() => {
        Console.WriteLine(Method2());
    });

    Task task3 = Task.Run(() => {
        Console.WriteLine(Method3());
    });
    Console.Read();
}
//Output: 20 30 10
```

Asynchronous
Programming

- ❖ The `async` keyword is used to mark a method as an asynchronous method.
- ❖ The `await` keyword is used to pause the execution of an asynchronous method until a specified operation completes.

```
static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

static int Method2(int i)
{
    return 20 * i;
}

static int Method3()
{
    return 30;
}

static void Main(string[] args)
{
    Method1_2();

    int k = Method3();
    Console.WriteLine(k);
    Console.Read();
}
```

```
//Asynchronous with Task async/await

static async void Method1_2()
{
    Console.WriteLine("Test");

    var i = await Task.Run(() =>
    {
        return Method1();
    });

    Console.WriteLine(i);

    int j = Method2(i);
    Console.WriteLine(j);
}

//Output: Test 30 10 200
```

Chapter 14 : SQL - Basics

Q120. What is the difference between DBMS and RDBMS?

Q121. What is a Constraint in SQL? What are the types of constraints? V Imp

Q122. What is the difference between Primary key and Unique key?

Q123. What are Triggers and types of triggers?

Q124. What is a View? V Imp

Q125. What is the difference between Having clause and Where clause? V Imp

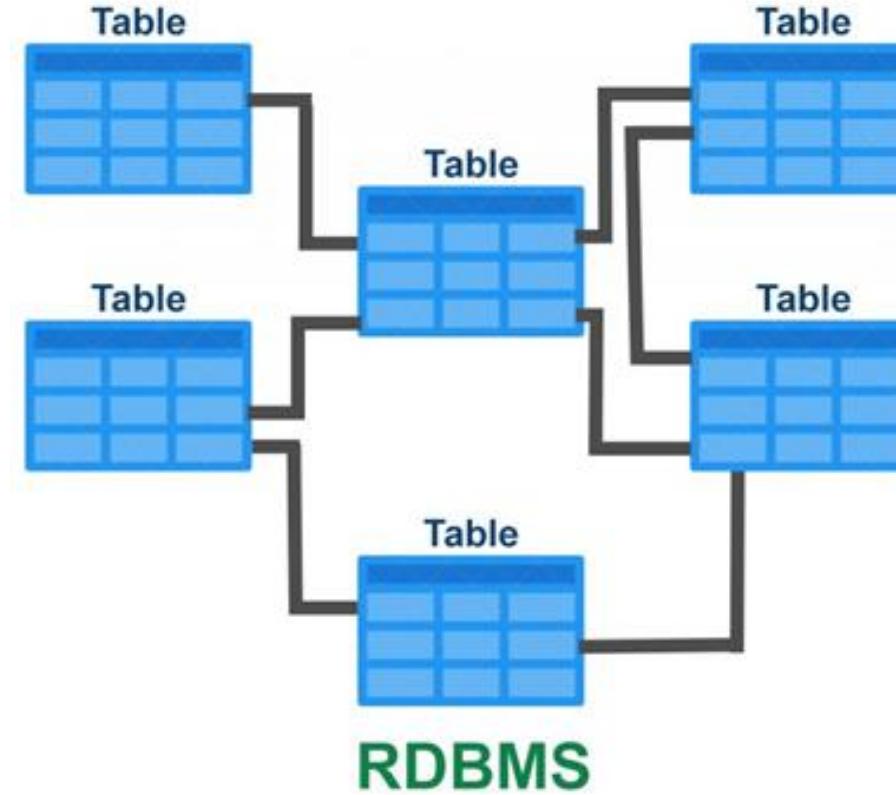
Q126. What is Sub query or Nested query or Inner query in SQL?

Q127. What is Auto Increment/ Identity column in SQL Server?





DBMS



DBMS	RDBMS
1. DBMS stores data as file.	RDBMS stores data in TABULAR form.
2. No relationship between data.	Data is stored in the form of tables which are RELATED to each other. Eg: Foreign key relationship.
3. Normalization is not present.	NORMALIZATION is present.
4. It deals with small quantity of data.	It deals with LARGE amount of data.
5. Examples: XML	Examples: MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc.

- ❖ SQL constraints are used to specify **rules** for the data in a table.

1. A **PRIMARY KEY** is a field which can uniquely identify each row in a table.

```
CREATE TABLE Students (
    ID int NOT NULL PRIMARY KEY,
    Name varchar(255) NOT NULL,
```

2. **NOT NULL** constraint tells that we cannot store a null value in a column.

3. A **FOREIGN KEY** is a field which can uniquely identify each row in another table.

```
CourseID int FOREIGN KEY REFERENCES Courses(CourseID),
    Age int NOT NULL CHECK (AGE >= 18),
    AdmissionDate date DEFAULT GETDATE(),
    CONSTRAINT UC_Student UNIQUE (ID,Name)
```

4. **CHECK** constraint helps to validate the values of a column to meet a particular condition.

6. **UNIQUE** constraint tells that all the values in the column must be unique.

5. **DEFAULT** constraint specifies a default value for the column when no value is specified by the user.

Primary Key

1. Primary key can't accept null values

2. Automatically create clustered index.

3. Only one primary key can be present in a table.

Unique Key

Unique key can accept only one null value

Create non clustered index.

More than one unique key can be present in a table.

- ❖ Triggers are stored programs, which are **AUTOMATICALLY** executed or fired when some events (insert, delete and update) occur.

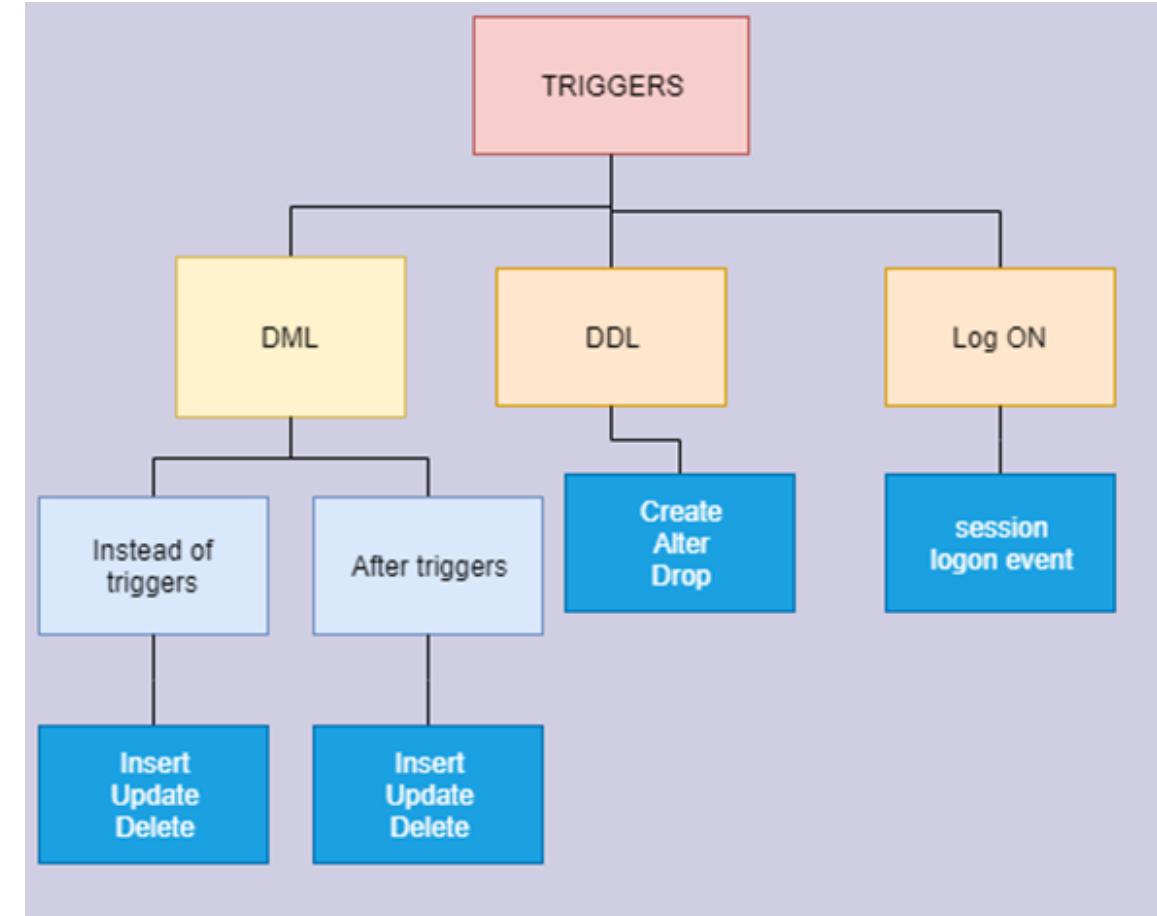
- ❖ Locations → LocationHist

- ❖ Example of After(DML) Trigger

```

CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
NOT FOR REPLICATION
AS
BEGIN
    INSERT INTO LocationHist
    SELECT LocationID
        ,getdate()
    FROM inserted
END
  
```

Table Name → Locations
 Trigger Name → TR_UPD_Locations
 DML Event → FOR UPDATE
 T-SQL block that runs against specified DML Event →
`INSERT INTO LocationHist
 SELECT LocationID
 ,getdate()
 FROM inserted`



- ❖ In after trigger, update on the table executed first and then after trigger will be fired.

- ❖ Triggers are stored programs, which are **AUTOMATICALLY** executed or fired when some events (insert, delete and update) occur.

- ❖ Example of Instead of(DML) Trigger

```

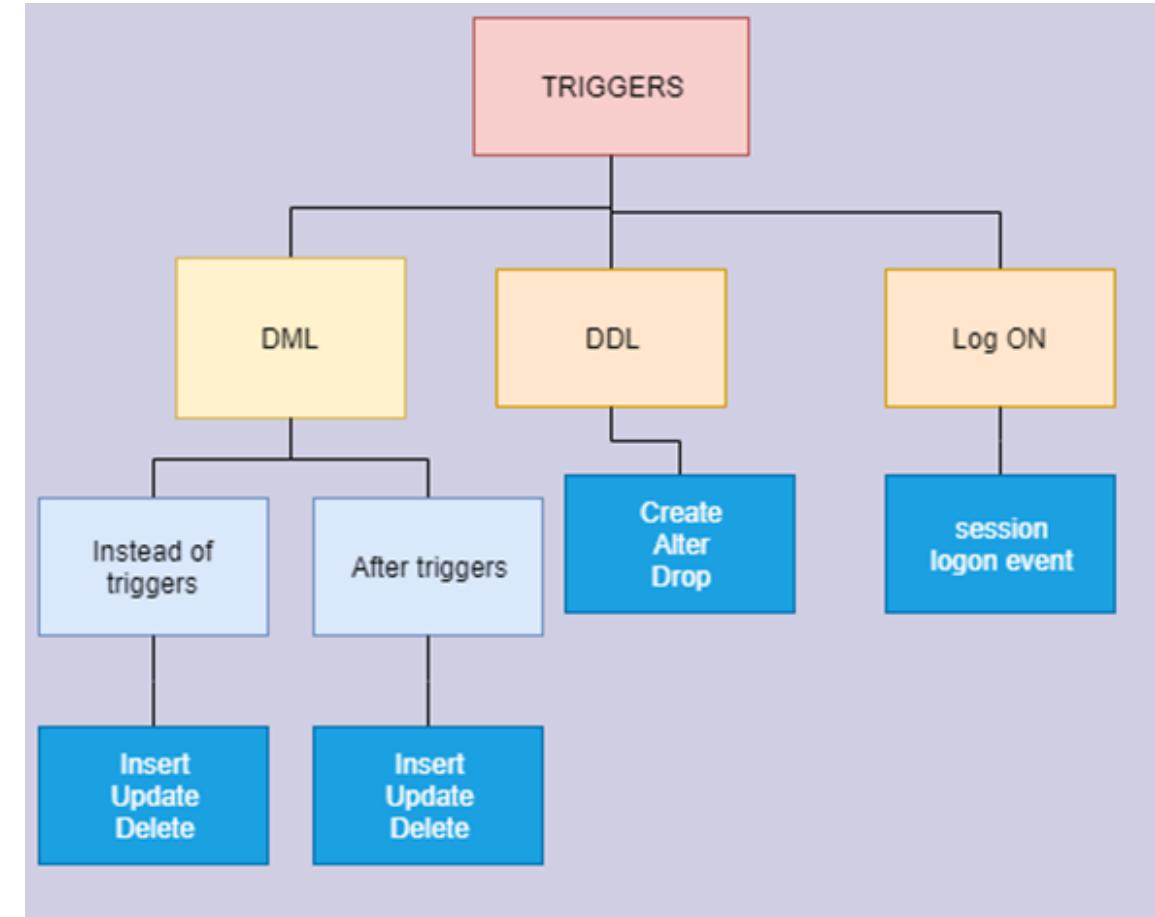
CREATE TRIGGER [dbo].[TRG_VM_EMPDETAILS]
ON [dbo].[vw_empdetails]
INSTEAD OF INSERT
AS
BEGIN
-- LOGIC HERE
END
  
```

Trigger Name: INSTEAD OF

View Name: vw_empdetails

INSTEAD OF Trigger

This trigger is only for INSERT



- ❖ An INSTEAD OF trigger is a trigger that allows you to **skip** an INSERT , DELETE , or UPDATE statement to a table or a view and execute other statements defined in the trigger.

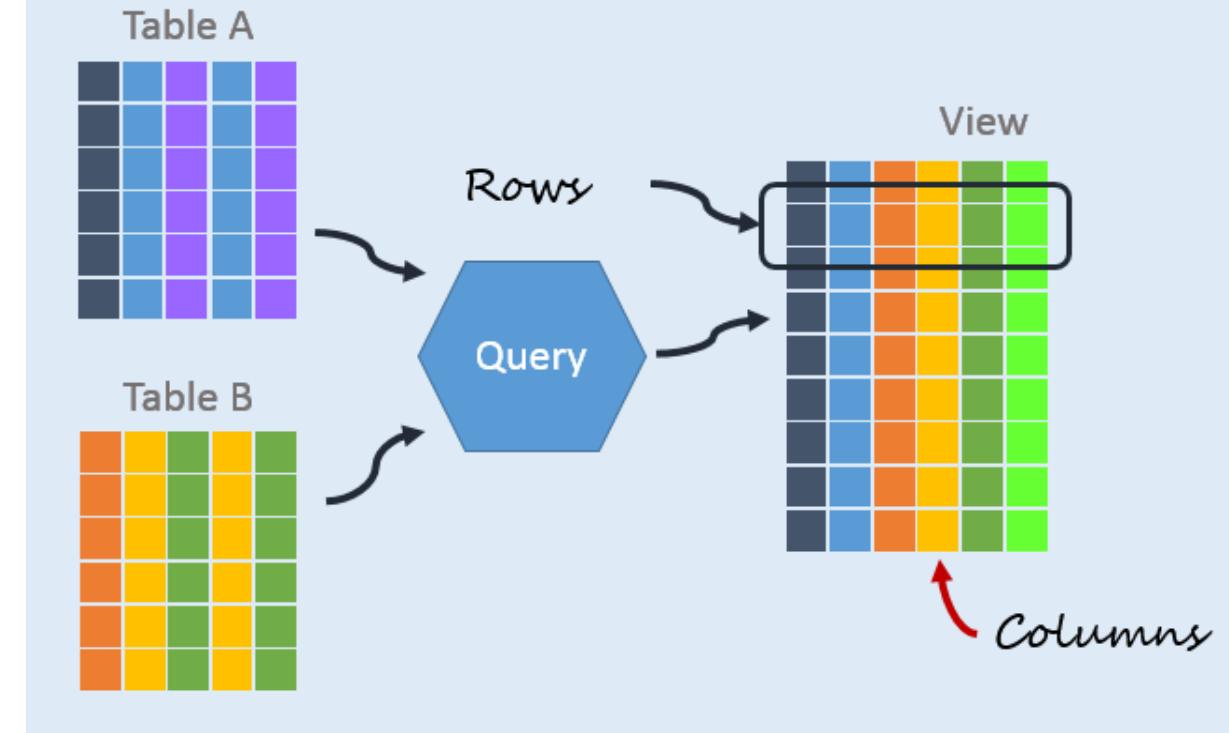
- ❖ A view is a VIRTUAL table which consists of a subset of data contained in single table or more than one table.

```
CREATE VIEW [India-Customers] AS
```

```
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'India';
```

- ❖ Advantages of Views

1. Indexed Views to improve the performance.
2. Extra security – DBA can hide the actual table names and expose views for Read operations only.



- ❖ Remember, in case of view, only query is stored but the actual data is never stored like a table.

1. WHERE Clause is used before GROUP BY Clause.

HAVING Clause is used after GROUP BY Clause.

```
SELECT COUNT(CustomerID), Country
FROM Customers
WHERE Country = "India"
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

2. WHERE Clause cannot contain AGGREGATE function.

HAVING Clause can contain aggregate function.

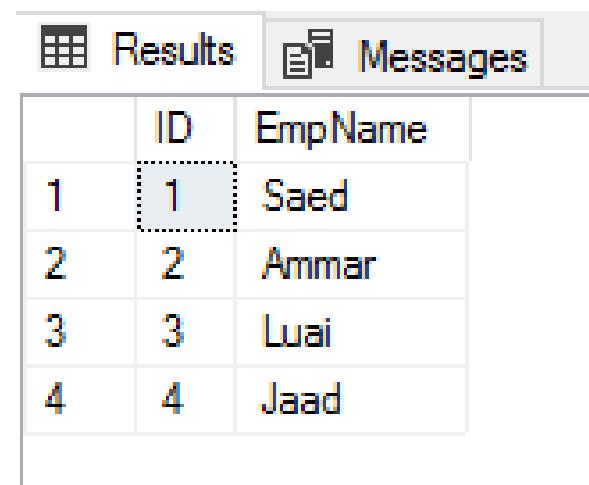
```
SELECT EmpName FROM Employee
GROUP BY EmpName
HAVING SUM(EmpSalary) < 30000
```

- ❖ A Subquery/ Inner query/ Nested query is a query within another SQL outer query and **embedded** within the **WHERE** clause.



- ❖ Auto-increment allows a unique number to be **generated automatically** when a new record is inserted into a table.
- ❖ Mostly auto increment is set on the primary key only.

```
CREATE TABLE Employee (
    ID int IDENTITY(1,1) PRIMARY KEY,
    EmpName varchar(255) NOT NULL
);
```



The screenshot shows the SSMS interface with the 'Results' tab selected. A table named 'Employee' is displayed with the following data:

	ID	EmpName
1	1	Saed
2	2	Ammar
3	3	Luai
4	4	Jaad

Chapter 15: SQL - Joins & Indexes

Q128. What are **Joins** in SQL?

Q129. What are the **types of Joins** in SQL Server? V Imp

Q130. What is **Self-Join**?

Q131. What are **Indexes** in SQL Server?

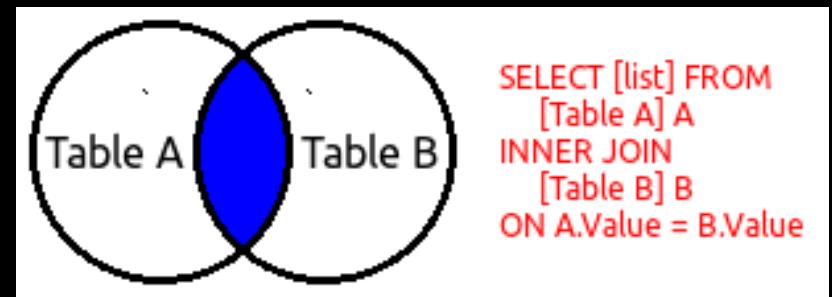
Q132. What is **Clustered** index?

Q133. What is **Non-Clustered** index?

Q134. What is the difference between Clustered and Non-Clustered index? V Imp

Q135. How to create Clustered and Non-Clustered index in a table?

Q136. In which column you will apply the indexing to optimize this query?

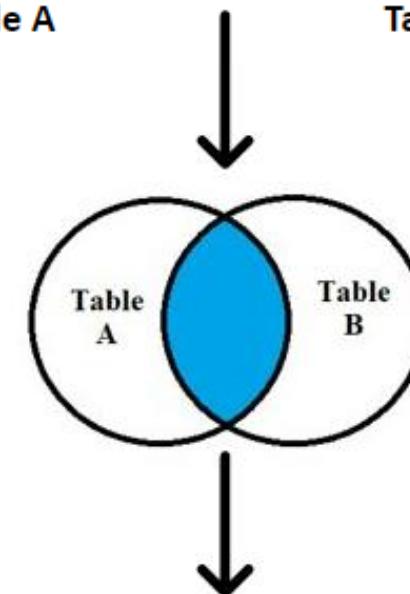


- ❖ A join clause is used to **combine** rows from two or more tables, based on a related column between them.

Student ID	Name	Student ID	Department
1001	A	1004	Mathematics
1002	B	1005	Mathematics
1003	C	1006	History
1004	D	1007	Physics
		1008	Computer Science

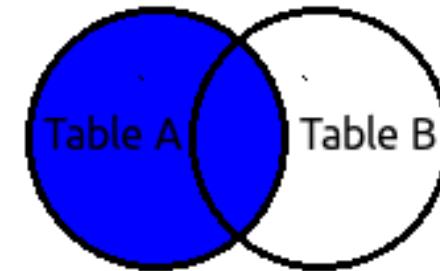
Table A

Table B



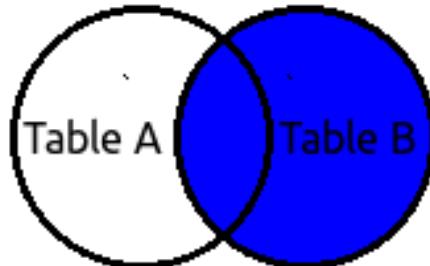
Student ID	Name	Department
1004	D	Mathematics

- ❖ **Left outer join** - A left join returns all the rows from the left table, along with any matching rows from the right table.



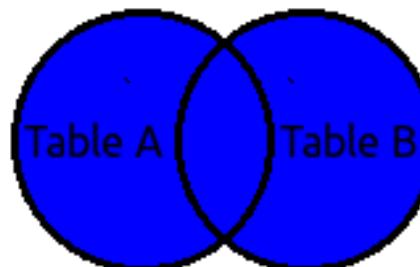
```
SELECT [list] FROM
[Table A] A
LEFT JOIN
[Table B] B
ON A.Value = B.Value
```

- ❖ **Right outer join** - A right join returns all the rows from the right table, along with any matching rows from the left table.



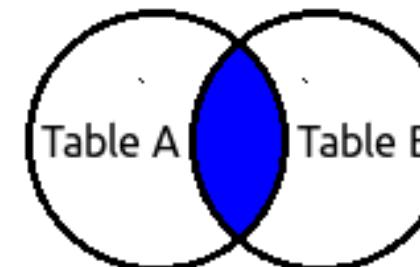
```
SELECT [list] FROM
[Table A] A
RIGHT JOIN
[Table B] B
ON A.Value = B.Value
```

- ❖ **Full outer join** - A full outer join returns all the rows from both the left and right tables in the join.



```
SELECT [list] FROM
[Table A] A
FULL OUTER JOIN
[Table B] B
ON A.Value = B.Value
```

- ❖ **Inner join** - An inner join returns only the common rows from both tables that meet the join condition.



```
SELECT [list] FROM
[Table A] A
INNER JOIN
[Table B] B
ON A.Value = B.Value
```

- ❖ A self join is a join of a table to **itself**.
- ❖ When to use Self Join??

employees	
*	employee_id
	first_name
	email
	phone_number
	hire_date
	job_id
	salary
	manager_id
	department_id

This manager id is
employee id of the
manager of an
employee

SELECT

e.first_name AS employee,
m.first_name AS manager

FROM

employees e LEFT JOIN
employees m
ON m.employee_id = e.manager_id
ORDER BY manager;

	employee	manager
▶	Steven King	NULL
	Bruce Ernst	Alexander Hunold
	David Austin	Alexander Hunold
	Valli Pataballa	Alexander Hunold
	Diana Lorentz	Alexander Hunold
	Alexander Khoo	Den Raphaely

- ❖ Now your task is to get the employees name with their manager names??

- ❖ SQL Indexes are used in relational databases to retrieve data **VERY FAST**.
- ❖ They are similar to indexes at the start of the BOOKS, which purpose is to find a topic quickly.

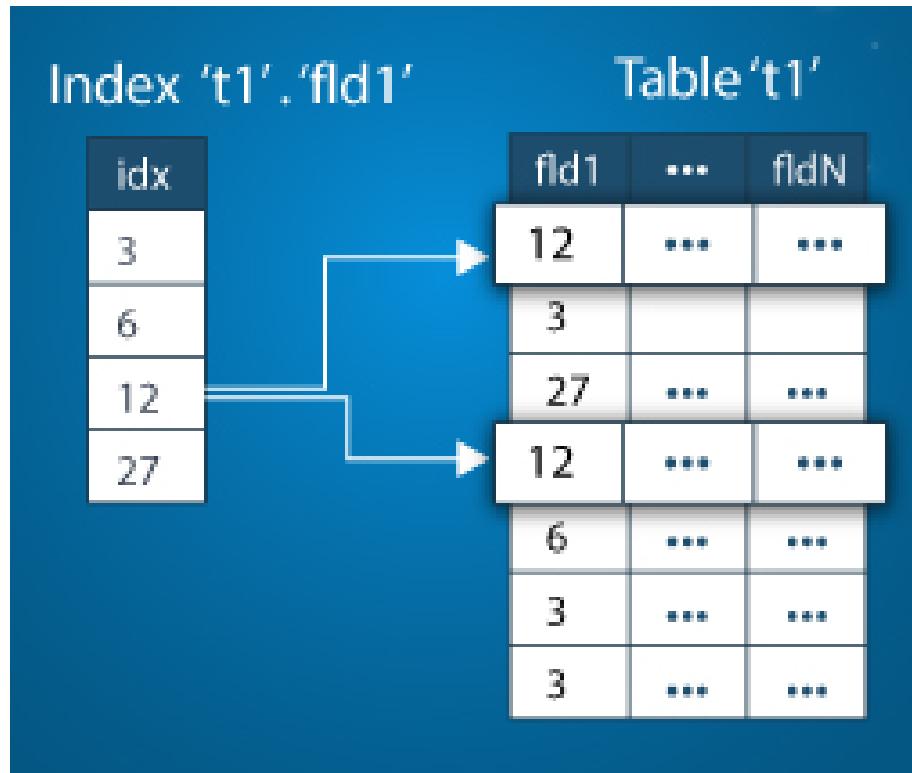
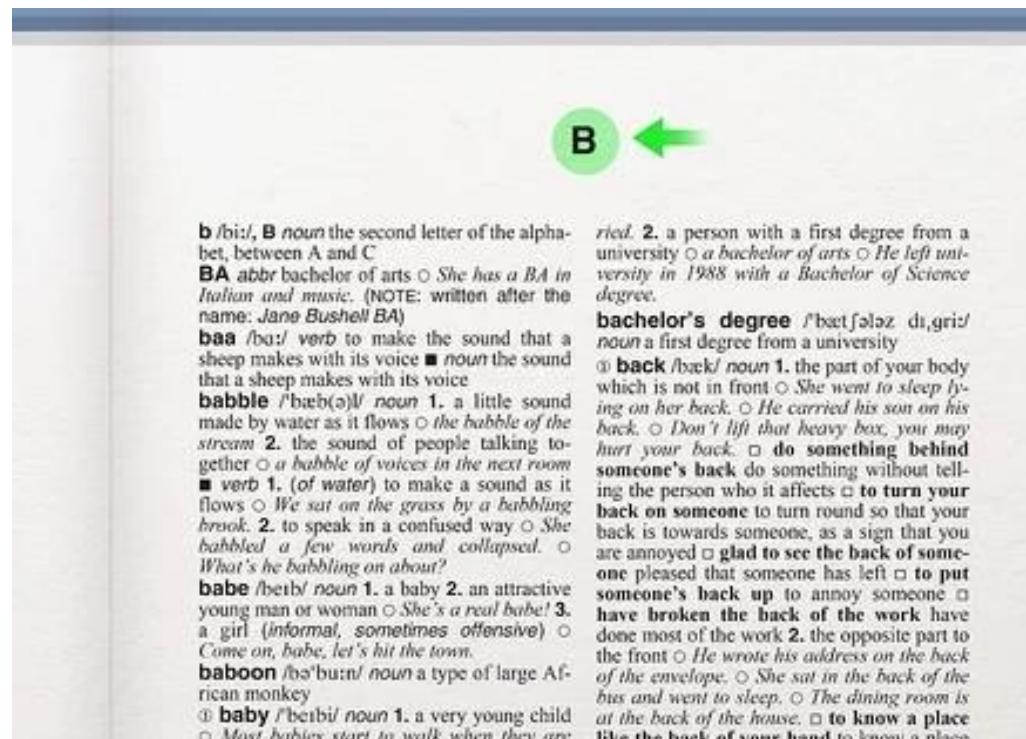


Table of Contents

Acknowledgments.....	ix
Introduction	xi
Part I Envision the Possibilities	
1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon.....	7

- ❖ A clustered index is a type of index that determines the **physical order** of data in a table.
- ❖ Table data can be sorted in only way, therefore, there can be **only one** clustered index per table.
- ❖ In SQL Server, if you set a primary key on a column, then it will **automatically create** a clustered index on that particular column.

Dictionary



- ❖ A non-clustered index is stored at one place and table data is stored in another place.
- ❖ A table can have multiple non-clustered index in a table.

Index

A

accordion, layouts
 about 128
 movie form, adding 131
 nesting, in tab 128, 129
 toolbar, adding 129-131
 adapters, Ext
 about 18
 using 18, 20
 Adobe AIR 285
 Adobe Integrated Run time. *See* Adobe AIR
 AJAX 12
 Asynchronous JavaScript and XML.
See AJAX

B

built-in features, Ext
 client-side sorting 86
 column, reordering 86, 87
 columns, hidden 86

lookup data stores, creating 83
 two columns, combining 84
 classes 254
 ComboBox, form
 about 47
 database-driven 47-50
 component config 59
 config object
 about 28, 29
 new way 28, 29
 old way 28
 tips 26, 29
 content, loading on menu item click 68, 69
 custom class, creating 256-259
 custom component, creating 264-266
 custom events, creating 262-264

D

data, filtering
 about 238
 remote, filtering 238-244

Book Index

Table of Contents

Acknowledgments	ix
Introduction	xi

Part I Envision the Possibilities

1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon	7

1. A clustered index is a type of index that determines the **physical order** of data in a table.

A non-clustered index is stored at one place and table data is stored in another place. For example, Book Index.

2. A table can have only one clustered index.

A table can have multiple non-clustered index.

3. Clustered index is faster.

Non-clustered index is slower.

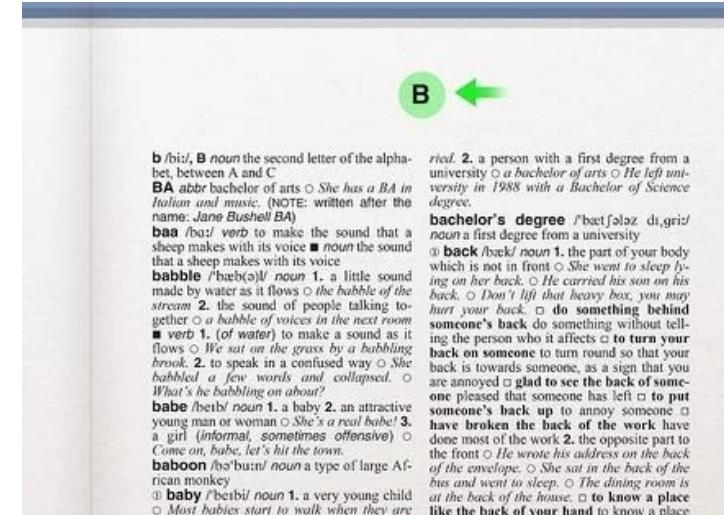


Table of Contents	
Acknowledgments	ix
Introduction	xi
Part I Envision the Possibilities	
1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon	7

❖ Clustered Index

```
CREATE CLUSTERED INDEX <index_name>
ON <table_name>(<column_name> ASC/DESC)
```

❖ Non-clustered Index

```
CREATE NONCLUSTERED INDEX <index_name>
ON <table_name>(<column_name> ASC/DESC)
```

- ❖ When you create a PRIMARY KEY constraint, a clustered index on the column is automatically created.

In which column you will apply the indexing to optimize this query.

“select id, class from student where name=“happy””?

select id, class from student where name=“happy”

- ❖ The column **after WHERE** condition, which is “NAME” here.

Chapter 16 : SQL - Stored Procedure, Functions & Others

Q137. What is the difference between **Stored Procedure** and **Functions**? V Imp

Q138. How to optimize a Stored Procedure or SQL Query?

Q139. What is a **Cursor**? Why to avoid them?

Q140. What is the difference between **scope_identity** and **@@identity**?

Q141. What is **CTE** in SQL Server?

Q142. What is the difference between **Delete**, **Truncate** and **Drop** commands? V Imp

Q143. How to get the **Nth highest salary** of an employee?

Q144. What are **ACID properties**?

Q145. What are **Magic Tables** in SQL Server?

```
CREATE PROCEDURE proc_name
(@Ename varchar(50),
@EId int output)
AS
BEGIN
    INSERT INTO Employee (EmpName)
    VALUES (@Ename)
    SELECT @EId= SCOPE_IDENTITY()
END
```

Stored Procedure	Function
1. SP may or may not return a value	Function must return a value
2. Can have input/output parameters	Only has input parameters
3. We can call function inside SP	Cannot call SP inside a function
4. We cannot use SP in SQL statements like SELECT, INSERT, UPDATE, DELETE, MERGE, etc.	We can use them with function. <i>SELECT *, dbo.fnCountry(city.long) FROM city;</i>
5. We can use try-catch exception handling in SP	We can not use try-catch in functions
6. We can use transactions inside SP.	We can not use transactions inside functions.

```
CREATE PROCEDURE proc_name
(@Ename varchar(50),
@EId int output)
AS
BEGIN
```

```
INSERT INTO Employee (EmpName)
VALUES (@Ename)
SELECT @EId= SCOPE_IDENTITY()
```

```
END
```

```
CREATE FUNCTION function_name
(parameters) --only input parameter
RETURNS data_type AS
BEGIN
  -- SQL statements
  RETURN value
```

```
END;
```

[back to chapter index](#)

1. Use SET NOCOUNT ON

2. Specify column names instead of using * .

~~SELECT * FROM table1~~

~~SELECT col1, col2 FROM table1~~

3. Use schema name before objects or table names.

~~SELECT EmpID, Name FROM Employee~~

~~SELECT EmpID, Name FROM dbo.Employee~~

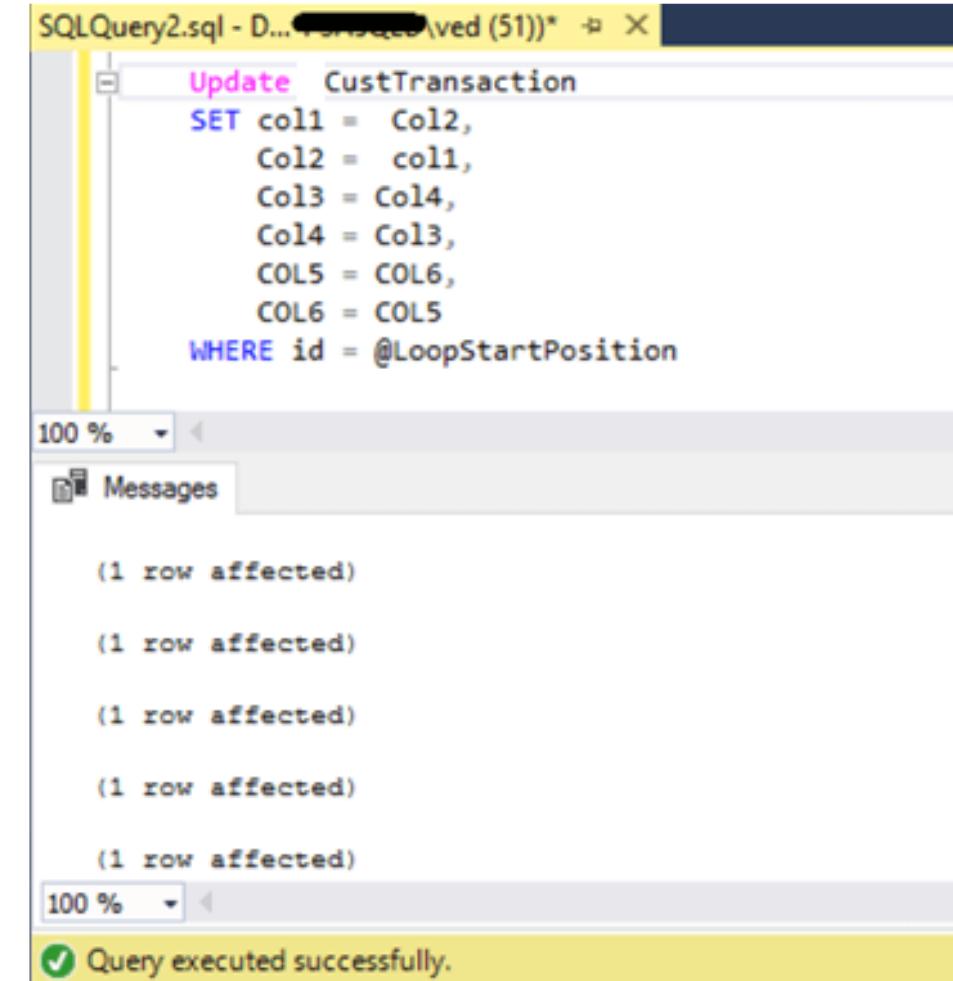
4. Use EXISTS () instead of COUNT () .

~~SELECT Count(1) FROM dbo.Employee~~

~~IF(EXISTS (SELECT 1 FROM db.Employees))~~

5. Use TRANSACTION when required only.

6. Do not use DYNAMIC QUERIES. They are vulnerable to SQL Injections.



The screenshot shows an SSMS window with the following content:

```
SQLQuery2.sql - D... [1] [Save] [New (51)] [X] |
```

Update CustTransaction

```
SET col1 = Col2,
    Col2 = col1,
    Col3 = Col4,
    Col4 = Col3,
    COL5 = COL6,
    COL6 = COL5
WHERE id = @LoopStartPosition
```

100 %

Messages

```
(1 row affected)
```

100 %

Query executed successfully.

- ❖ A database Cursor is a control which enables traversal/ iteration over the rows or records in the table.

- ❖ 5 step process:

1. Declare
2. Open
3. Fetch using while loop
4. Close
5. Deallocate

- ❖ LIMITATION

A cursor is a MEMORY resident set of pointers. Meaning it occupies lots of memory from your system which is not good for performance.

```

DECLARE
    @product_name VARCHAR(MAX),
    @list_price    DECIMAL;

DECLARE cursor_product CURSOR
FOR SELECT
    product_name,
    list_price
    FROM
        OPEN cursor_product;
        FETCH NEXT FROM cursor_product INTO
            @product_name,
            @list_price;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            PRINT @product_name + CAST(@list_price AS varchar);
            FETCH NEXT FROM cursor_product INTO
                @product_name,
                @list_price;
        CLOSE cursor_product;
        DEALLOCATE cursor_product;
    
```

- ❖ Both are used to get the last value generated in the **identity** column of the table.
- ❖ `@@IDENTITY` function returns the last identity value generated within the **current session**, regardless of the scope.
- ❖ This will return a value generated by an `INSERT` statement in a **trigger, stored procedure or batch** of T-SQL statements.

```
SELECT @@IDENTITY
```

- ❖ The `scope_identity()` function returns the last identity created in the same session and the same **scope**.

```
SELECT SCOPE_IDENTITY()
```

- ❖ Mostly we use `scope_identity()` function inside stored procedures.

- ❖ A Common Table Expression, is a TEMPORARY named result set, that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

```
WITH  
with engineers as (  
    select *  
    from employees  
    where dept='Engineering'  
)  
  
select *  
from engineers ← CTE Usage  
where ...
```

Diagram annotations:

- WITH**: Points to the keyword **WITH** in the first line of the CTE definition.
- CTE name**: Points to the alias **engineers** in the **as** clause.
- CTE Body**: Points to the **select** statement and the **from** clause of the CTE definition.
- CTE Usage**: Points to the **from** clause of the outer query, which is referencing the CTE.

DELETE	TRUNCATE	DROP
1. It is a DML.	1. It is a DDL.	1. It is a DDL.
2. It is used to delete one or all rows from the table based on where condition, but it will not delete schema.	2. It is used to delete all rows from the table, but it will not delete schema.	2. It is used to delete all rows from the table with structure/schema.
3. It can be rollback.	3. It can not be rollback.	3. It can not be rollback.

```
DELETE FROM Employees  
WHERE Emp_Id = 7;
```

```
TRUNCATE TABLE Employees;
```

```
DROP TABLE Employees;
```

1. The logic is first select TOP 3 salaries in descending order.

```
SELECT DISTINCT TOP 3 SALARY
  FROM tbl_Employees
 ORDER BY SALARY DESC
```

Salary
5000
10000
6000
4000
2000
7000

10000
7000
6000

2. Put the result in “Result” and then do order by asc

```
SELECT SALARY
  FROM (
```

```
SELECT DISTINCT TOP 3 SALARY
  FROM tbl_Employees
 ORDER BY SALARY DESC
```

```
) RESULT
 ORDER BY SALARY
```

Result
6000
7000
10000

3. Select top 1 salary from result set

```
SELECT TOP 1 SALARY
  FROM (
```

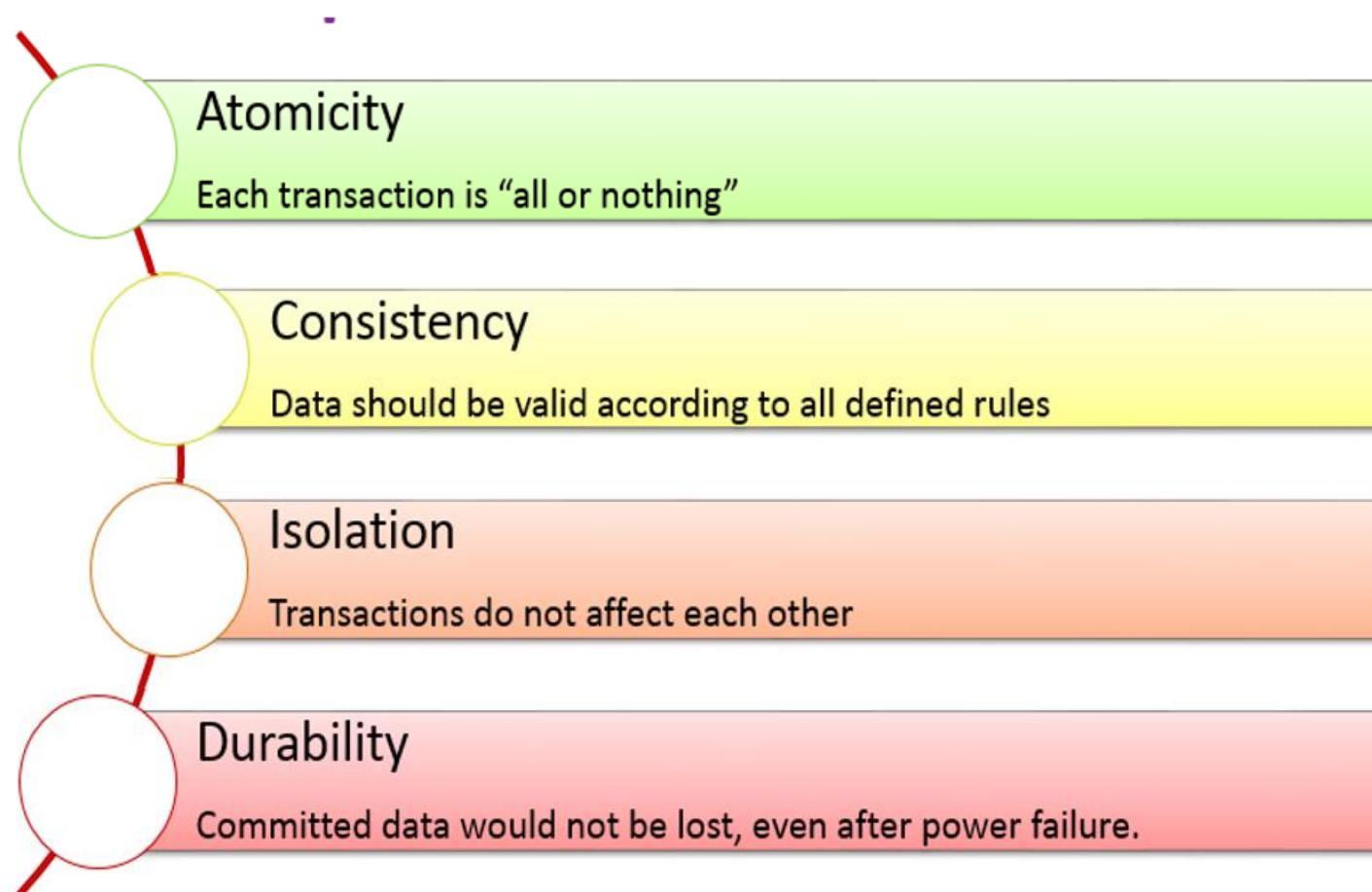
```
SELECT DISTINCT TOP 3 SALARY
  FROM tbl_Employees
 ORDER BY SALARY DESC
```

```
) RESULT
 ORDER BY SALARY
```

Result
6000

- ❖ ACID properties are used when you are handling **transactions** in SQL.

For example, multiple inserts in a table are happening at the same point of time.



- ❖ **Magic tables** are the temporary logical tables that are created by the SQL server, whenever there are **insertion or deletion or update**(D.M.L) operations.

- ❖ Types of magic tables

 1. **INSERTED** – When any insert query executed, then the recently inserted row gets added to the INSERTED magic table.

 2. **DELETED** – When any delete query executed, then the recently deleted row gets added to the DELETED magic table.

In update case, the updated row gets stored in INSERTED magic table and the old row or previous row gets stored in the DELETED magic table.

- ❖ The use of magic tables are TRIGGERS.

Chapter 17 : ASP.NET – MVC - Part 1

Q146. What is **MVC**? Explain MVC Life cycle.

Q147. What are the **advantages of MVC** over Web Forms? V Imp

Q148. What are the different **return types** of a controller Action method? V Imp

Q149. What are **Filters** and their types in MVC? V Imp

Q150. What is **Authentication** and **Authorization** in ASP.NET MVC? V Imp

Q151. What are the **types of Authentication** in ASP.NET MVC?

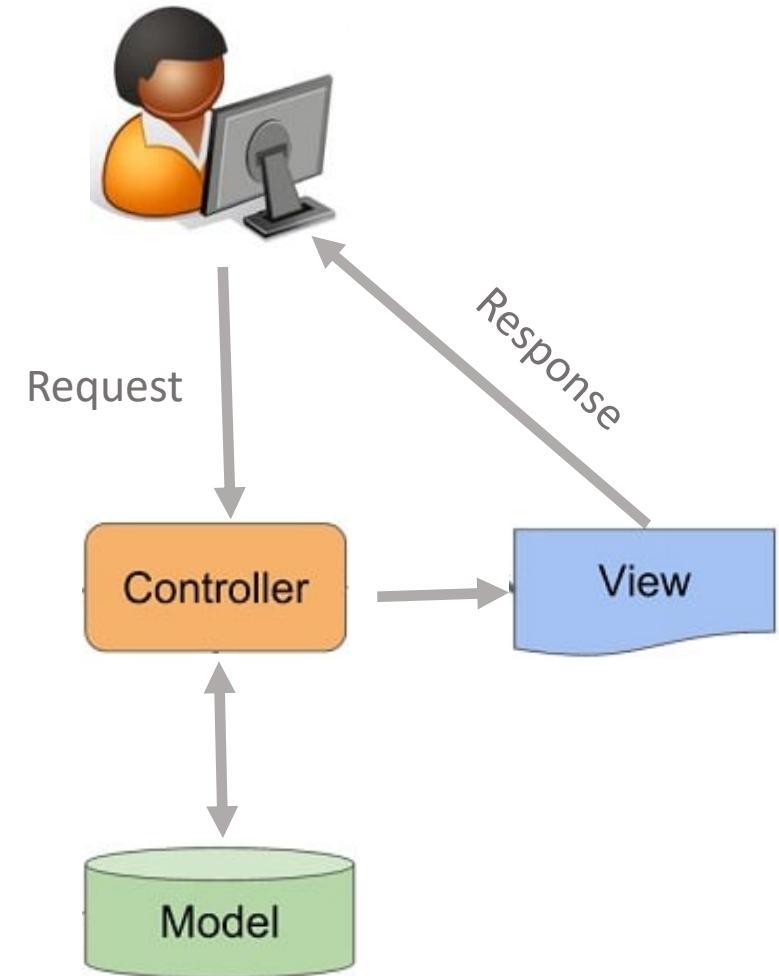
Q152. What is **Output Caching** in MVC? How to implement it?

Q153. What is **Routing** in MVC?

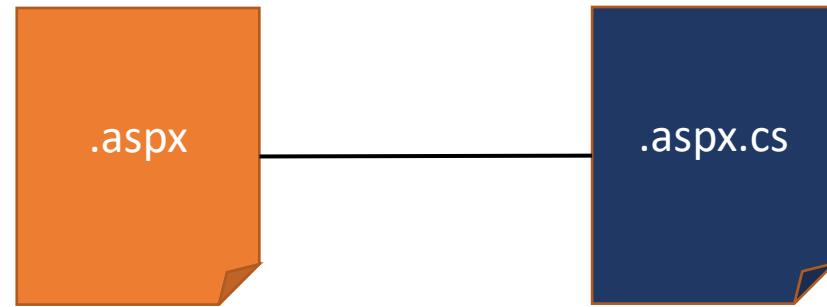
Q154. Explain **Attribute Based Routing** in MVC? V Imp



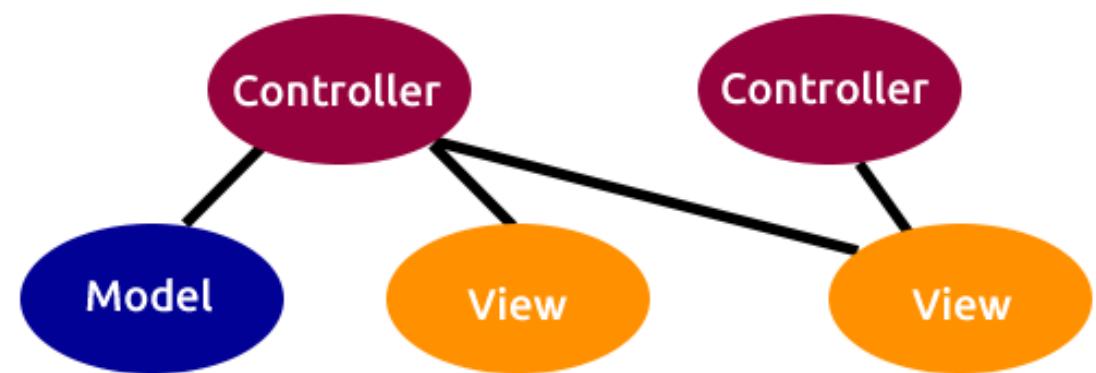
- ❖ MVC is a framework for building web applications using MVC (Model View Controller) architecture.
 1. The **Model** represents the data.
 2. The **View** displays the data.
 3. The **Controllers** act as an interface between Model and View components to process all the business logic.



- ❖ In web forms one aspx file will have one aspx.cs file, which means UI(aspx) is **tightly coupled** with logic in code-behind (.aspx.cs).

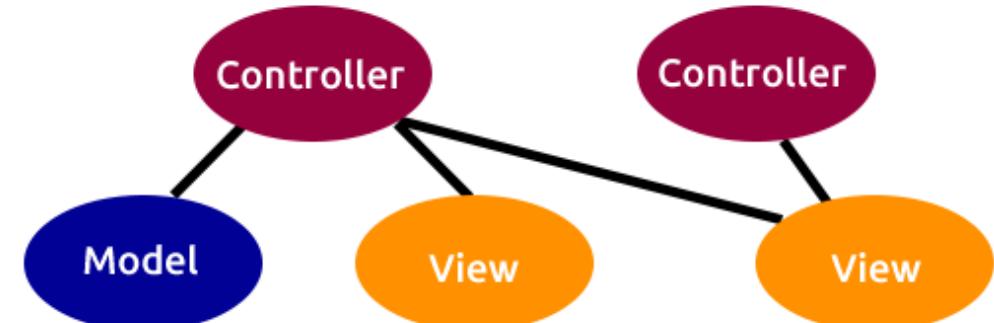


- ❖ In MVC, one controller can interact with multiple views and one view can interact with multiple controllers and therefore they are **loosely coupled**.



Advantage of MVC over Web-Forms:

- 1. Separation of concerns** - The MVC framework provides a clean separation of the UI, business logic and data with the help of MVC architecture.
- 2. Multiple view support** - Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.
- 3. Change accommodation** - UI change more frequently than business rules. Now because the model does not depend on the views, modifying view will not affect the model.
- 4. Testability** - ASP.NET MVC provides better support for test driven development.
- 5. Light-weight** – In MVC framework, the request and response are bit lighter than webforms.
- 6. Full features of Asp.Net** – Almost all the features of web-forms are present in ASP.NET MVC.

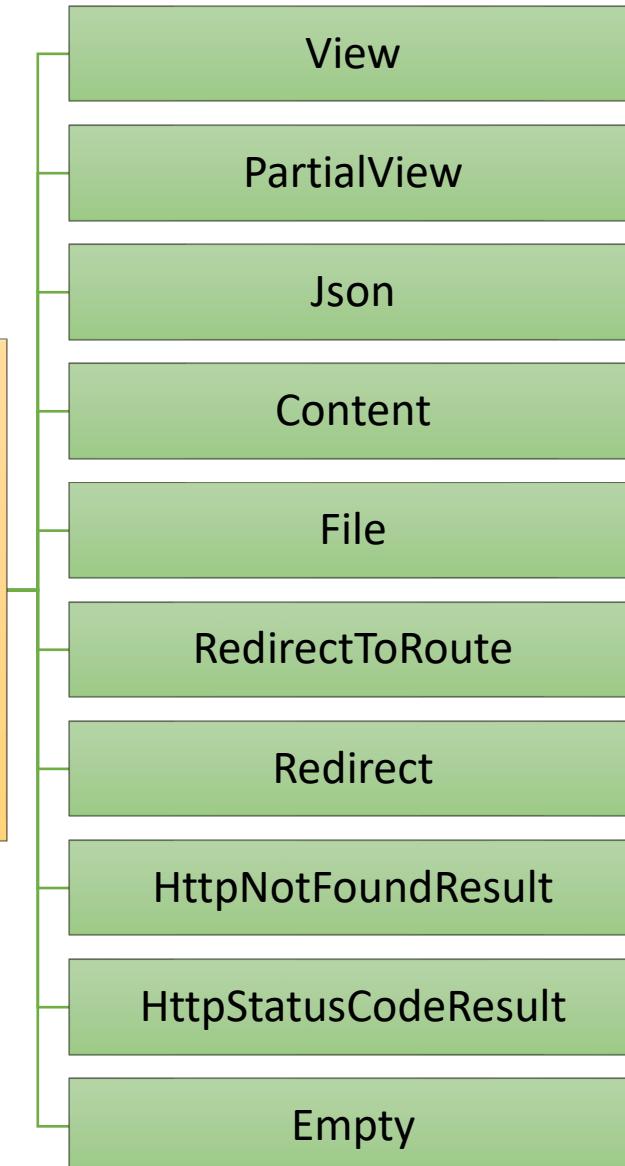


```
public ViewResult Index()
{
    // Do some processing to get data
    var data = SomeService.GetData();

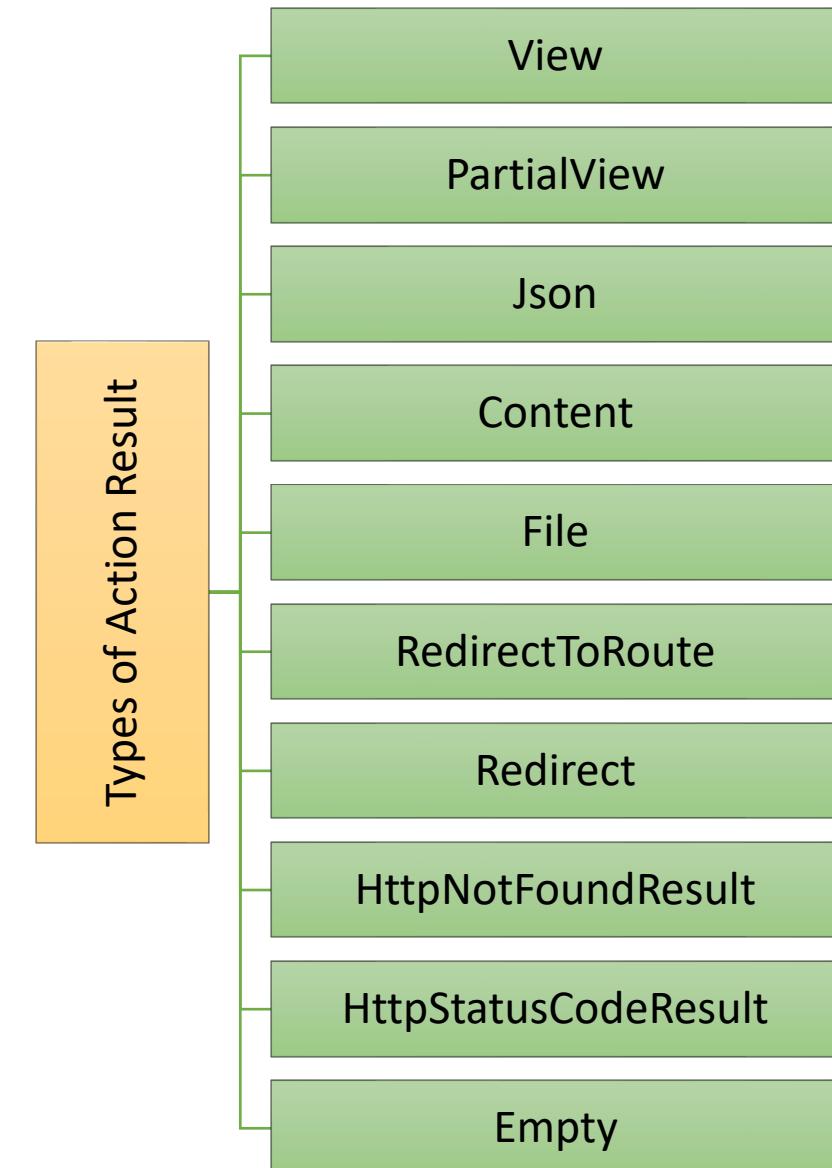
    return View(data);
}
```

- ViewResult:** It returns a view that renders HTML to be sent to the client.
- PartialViewResult:** It returns a partial view that is rendered within another view.
- JsonResult:** It returns JSON data that can be consumed by client-side JavaScript code.

Types of Action Result

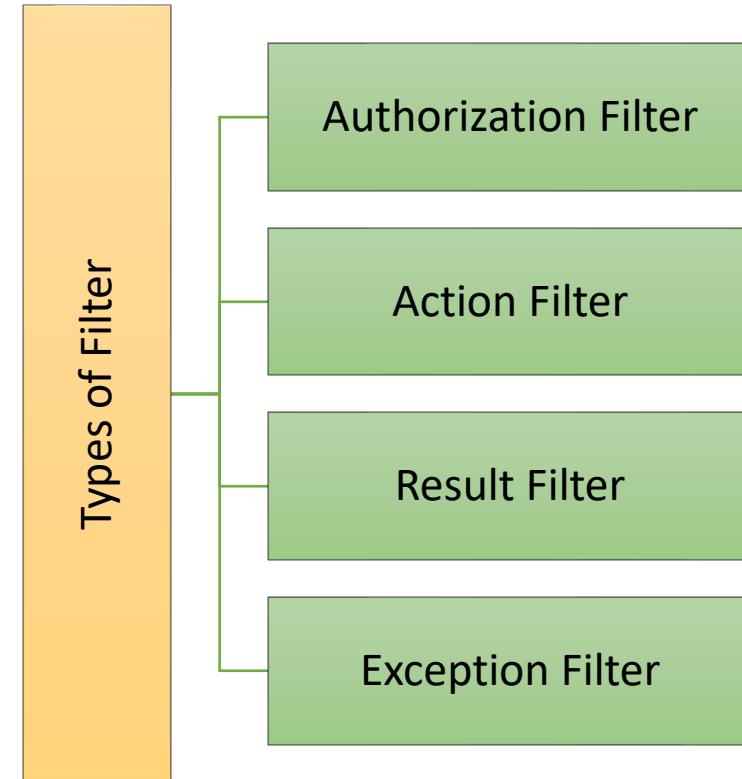


4. **ContentResult**: It returns a string of content that is directly written to the response stream. This is useful for returning plain text, XML, or other non-HTML content.
5. **FileResult**: It returns a file to the client for download, such as a PDF, image, or video.
6. **RedirectToRouteResult**: It redirects the client to another action method based on a specified route.
7. **RedirectResult**: It redirects the client to a specified URL.
8. **HttpNotFoundResult**: It returns a 404 Not Found status code to the client.
9. **HttpStatusCodeResult**: It returns an HTTP status code to the client, such as 401 Unauthorized or 500 Internal Server Error.
10. **EmptyResult**: It returns an empty response to the client without any content.



- ❖ Filters in ASP.NET MVC are used to inject logic that runs before or after an action method is executed.

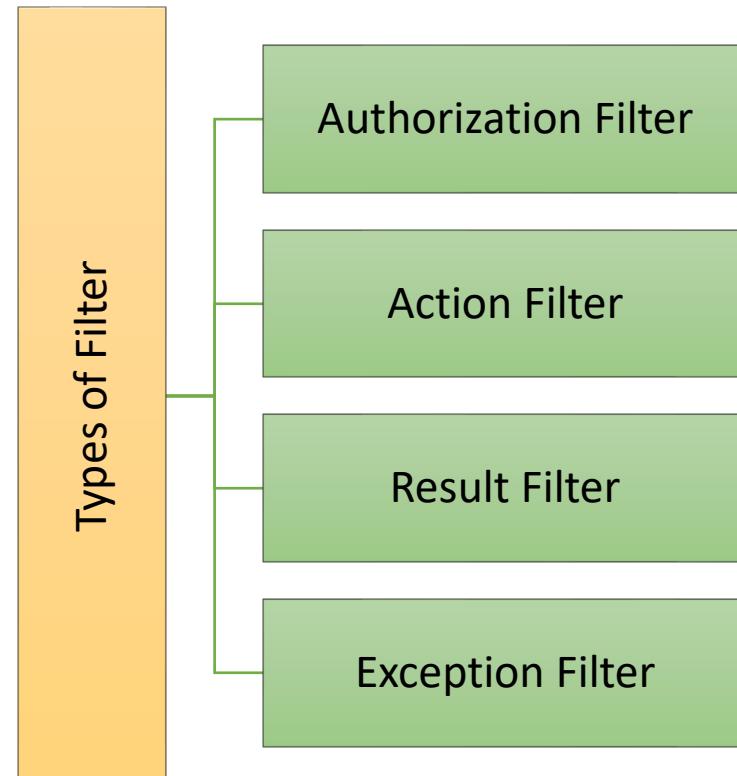
```
[Authorize(Roles = "Admin")]
public ActionResult Create()
{
    // Logic for creating a new
}
```



- 1. Authorization Filters:** Authorization filters are used to enforce authentication and authorization rules.

They run before the action method is executed and can be used to verify whether the user is authenticated and authorized to perform the requested action.

```
[Authorize(Roles = "Admin")]
public ActionResult Create()
{
    // Logic for creating a new
}
```



2. Action Filters: Action filters run before and after an action method is executed. They can be used to modify the behavior of the action method or perform pre-processing or post-processing tasks.

```
[LogActionFilter]  
  
public ActionResult Index()  
{  
    // Logic for displaying a  
}
```

```
public class LogActionFilter : ActionFilterAttribute  
{  
    public override void OnActionExecuting(ActionExecutingContext filterContext)  
    {  
        // This code will be executed before the action method is called  
    }  
  
    public override void OnActionExecuted(ActionExecutedContext filterContext)  
    {  
        // This code will be executed after the action method is called  
    }  
}
```

3. Result Filters: Result filters run before and after the result of an action method is executed. They can be used to modify the behavior of the result or perform post-processing tasks.

```
[OutputCache(Duration = 3600)]  
[LogResultFilter]  
public ActionResult Index()  
{  
    // Your action logic here  
    return View();  
}
```

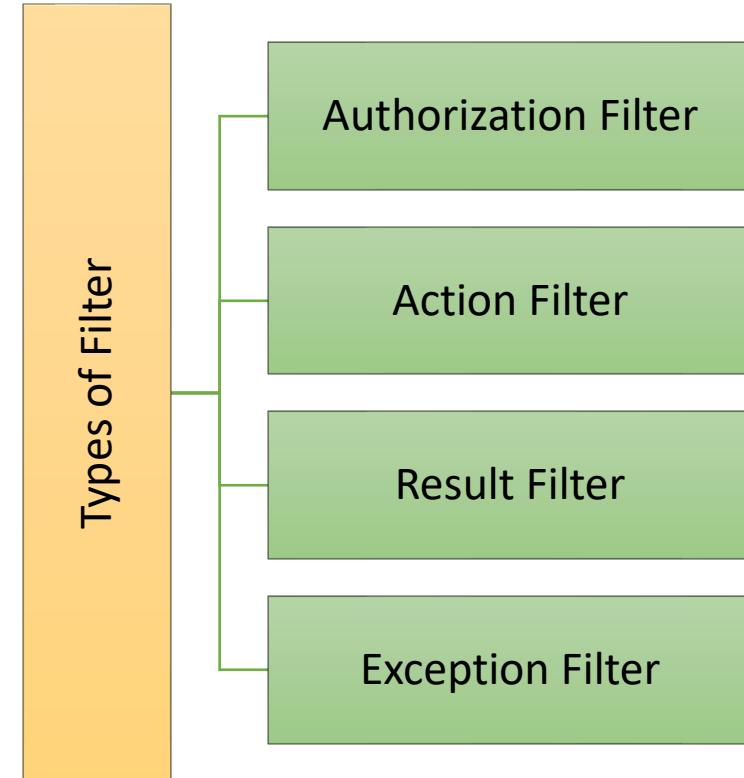
```
public class LogResultFilter : FilterAttribute, IResultFilter  
{  
    public void OnResultExecuting(ResultExecutingContext filterContext)  
    {  
        // This code will be executed before the action result  
    }  
  
    public void OnResultExecuted(ResultExecutedContext filterContext)  
    {  
        // This code will be executed after the action result  
    }  
}
```

4. Exception Filters: Exception filters are used to handle exceptions that occur during the execution of an action method. They run only when an unhandled exception occurs.

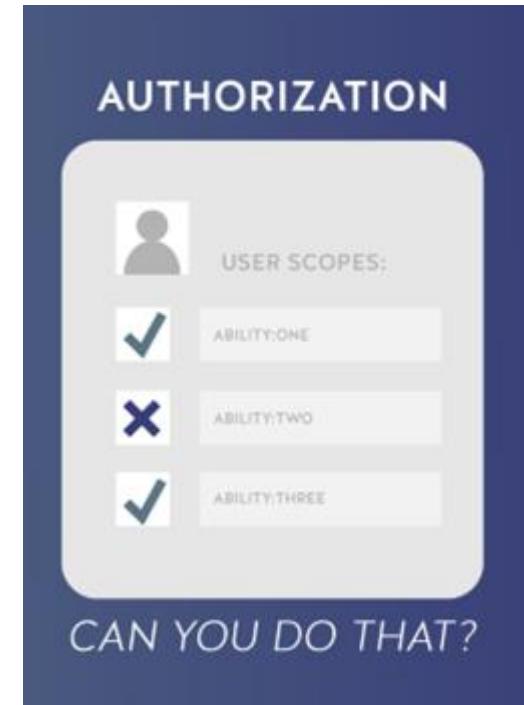
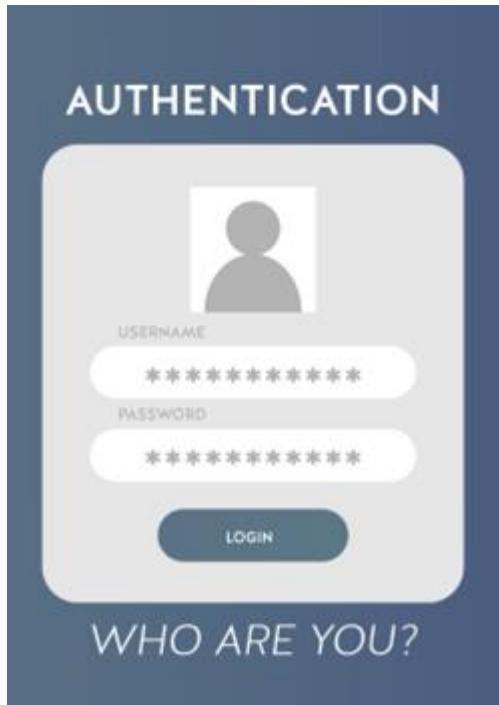
```
[LogExceptionFilter]
public ActionResult Index()
{
    // Logic for displaying
}
```

```
public class LogExceptionFilter : FilterAttribute, IExceptionFilter
{
    public void OnException(ExceptionContext filterContext)
    {
        // This code will be executed when an exception is thrown in
    }
}
```

- ❖ Filters in ASP.NET MVC are used to inject logic that runs before or after an action method is executed.



- ❖ Authentication is the process of verifying the identity of a user by validating their credentials such as username and password.
- ❖ Authorization is the process of allowing an authenticated user **ACCESS** to resources. Authentication is always precedes to Authorization.



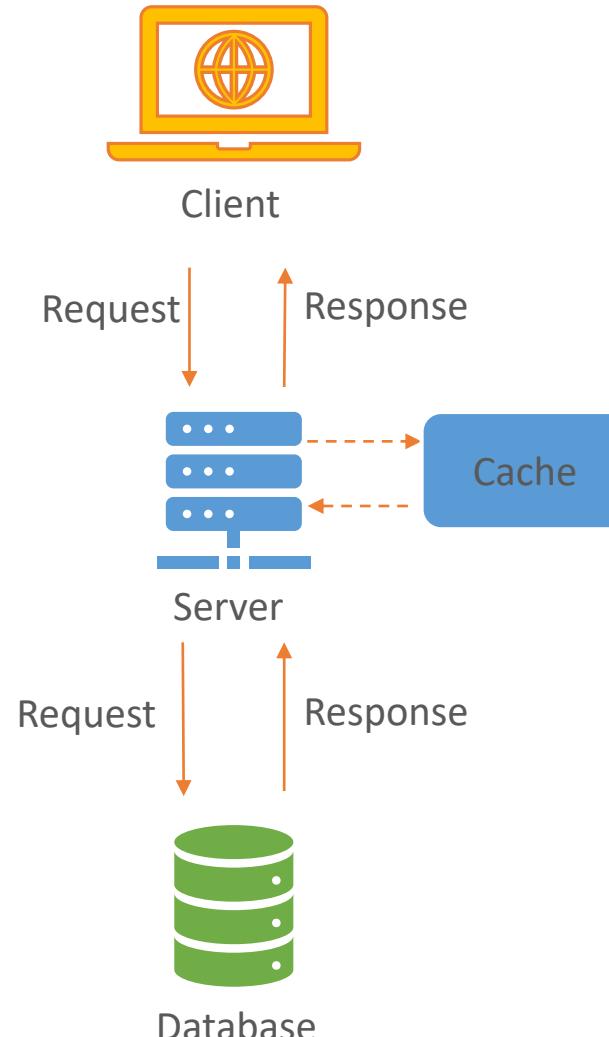
- 1. Form Authentication** - Forms authentication is a cookie-based authentication mechanism that uses user credentials stored in a database to authenticate users.
- 2. Passport Authentication** - Passport authentication is a centralized authentication service provided by Microsoft.
- 3. Token Authentication:** Token authentication is a mechanism that uses JSON Web Tokens (JWT) to authenticate users. It is commonly used in Single Page Applications (SPA) or APIs where the user's credentials are not stored on the server.
- 4. Windows Authentication** - Windows authentication is a mechanism that uses the user's Windows credentials to authenticate users. It is typically used in intranet scenarios where all users are part of the same Windows domain.

```
<authentication mode="Forms">
  <forms loginUrl="Accounts/Login"></forms>
</authentication>
```



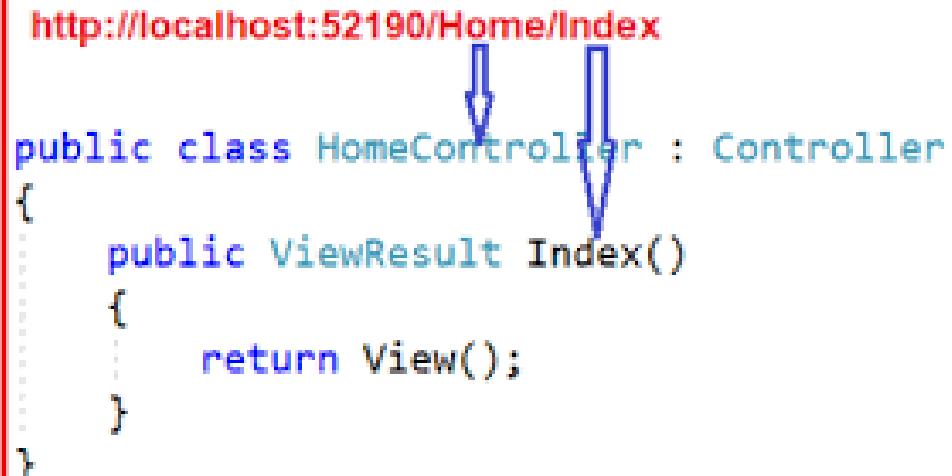
- ❖ Output caching in MVC is a technique used to improve the performance of web applications by storing the output of an action method in cache.

```
[OutputCache(Duration = 3600)]  
public ActionResult Index()  
{  
    return View();  
}
```



- ❖ Outputcache is one of the type of ResultFilter only

- ❖ Routing in MVC is the process of mapping a URL request to a specific controller action in a web application.
- ❖ The routing system is responsible for directing incoming requests to the appropriate controller, based on the URL.
- ❖ The routing system is typically configured using a **routing table**, which maps URLs to controller actions.



http://localhost:52190/Home/Index

```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        return View();
    }
}
```

- ❖ Attribute based routing is used to manipulate the default behavior of routing in ASP.NET MVC.

`http://localhost:1234/home/about`

```
public class HomeController: Controller
{
    [Route("Users/about")]
    Public ActionResult GotoAbout()
    {
        return View();
    }
}
```

Chapter 18 : ASP.NET – MVC - Part 2

Q155. What is the difference between **ViewData**, **ViewBag** & **TempData**? V Imp

Q156. How can we pass the data from controller to view in MVC?

Q157. What is **Partial View**?

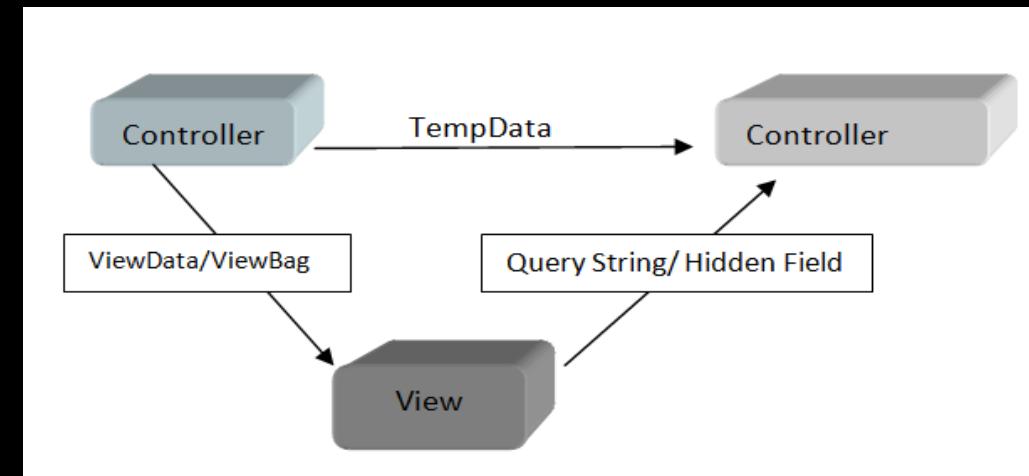
Q158. What are Areas in MVC?

Q159. How Validation works in MVC? What is **Data Annotation**?

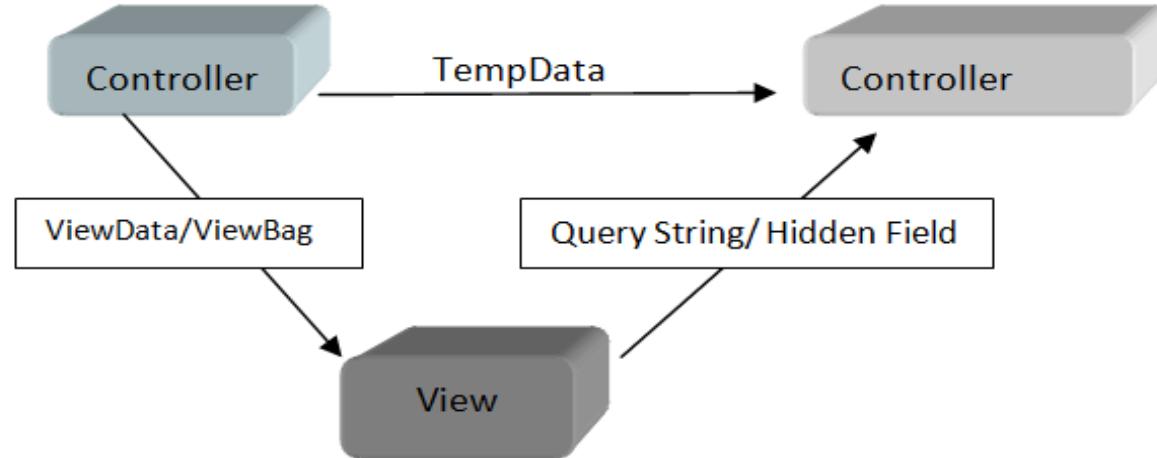
Q160. Explain the concept of MVC **Scaffolding**?

Q161. What is **Bundling** and **Minification** in MVC?

Q162. How to implement Security in web applications in MVC? V Imp



- ❖ ViewData, ViewBag, and TempData are all ways to **pass data** between a controller and a view.
1. ViewData and ViewBag are used to transfer data from controller to view.
 2. TempData is used to pass data from controller to controller(pass data between two action methods).
 3. ViewBag doesn't require typecasting, whereas ViewData require the typecasting.



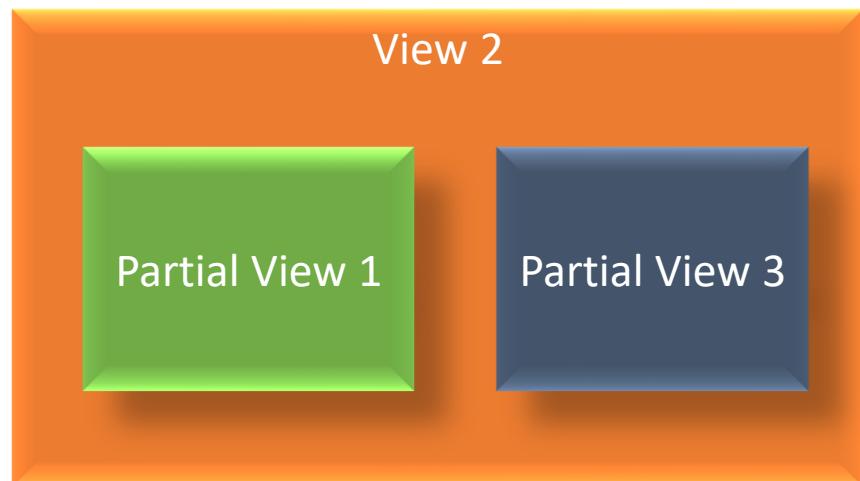
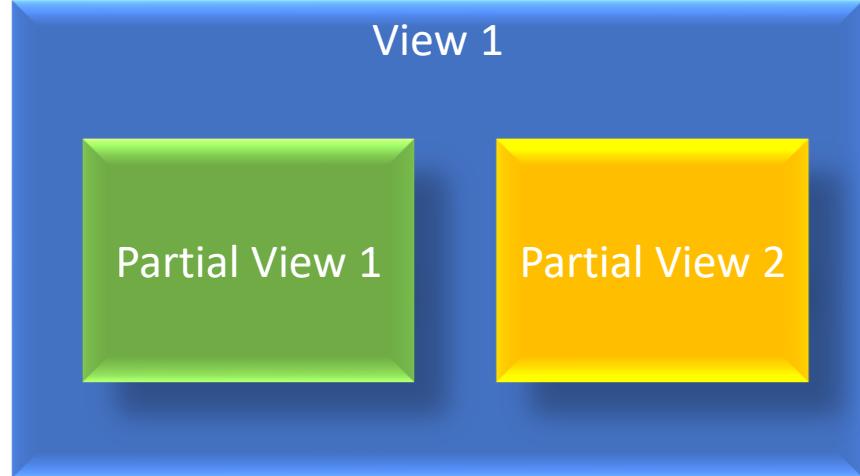
```
public ActionResult Index()
{
    ViewBag.Number = 42;
    ViewData["Number"] = 42;
    return View();
}
```

```
<p>Using ViewBag: @(ViewBag.Number)</p>
<p>Using ViewData: @((int)ViewData["Number"])</p>
```

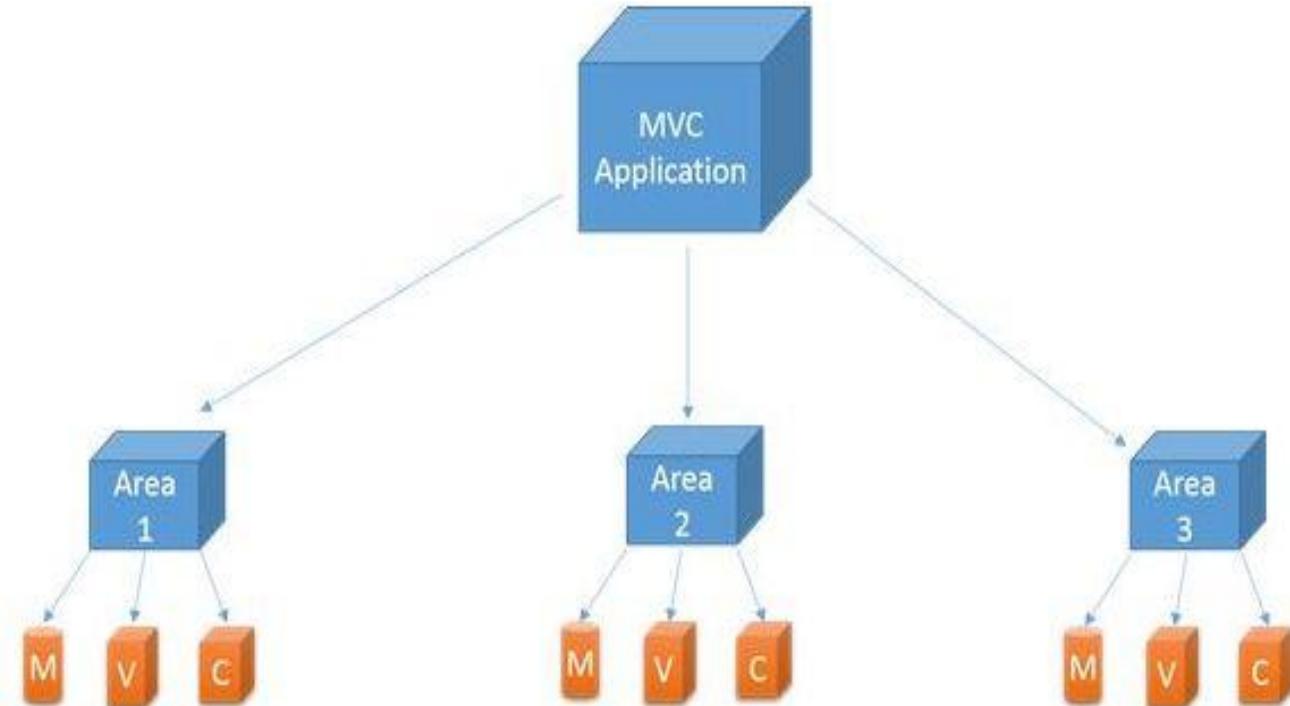
- ❖ By using ViewData and ViewBag

- ❖ A partial view is a reusable view component that can be rendered within other views or shared across multiple pages in a web application.
- ❖ It helps us to reduce code duplication.

```
<div class="container">
    <h1>Products</h1>
    @Html.Partial("_ProductList")
</div>
```



- ❖ Areas are just a way to divide the modules of large applications in multiple or separated MVC.



- ❖ Validation can be done in MVC using **DATA ANNOTATION** Attributes.

```
public class Student
{
    public int StudentId { get; set; }

    [Required]
    public string StudentName { get; set; }

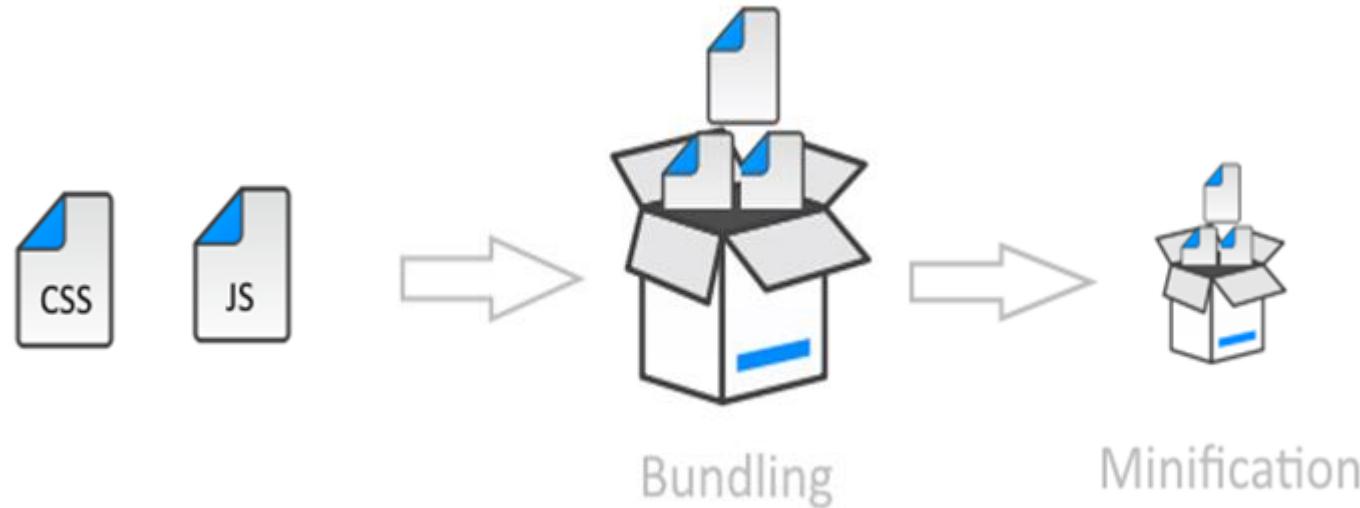
    [Range(10, 20)]
    public int Age { get; set; }
}
```

- ❖ [Required]: Specifies that a property is required and cannot be null or empty.
- ❖ [StringLength]: Specifies the minimum and maximum length of a string property.
- ❖ [RegularExpression]: Specifies a regular expression pattern that a string property must match.
- ❖ [EmailAddress]: Specifies that a string property must be a valid email address format.
- ❖ [Range]: Specifies the minimum and maximum allowed values for a numeric property.
- ❖ [Compare]: Compares the value of a property with the value of another property in the same model.
- ❖ [DataType]: Specifies the data type of a property, which can be used for formatting and validation purposes.

- ❖ [Display]: Provides metadata for displaying the property, such as the name and order of the property in a view.
- ❖ [Editable]: Specifies whether a property can be edited or displayed in a view.
- ❖ [ReadOnly]: Specifies that a property can be displayed in a view but cannot be edited.
- ❖ [HiddenInput]: Specifies that a property should be hidden in a view.
- ❖ [DisplayName]: Provides a friendly name for a property that can be used in a view.
- ❖ [DisplayFormat]: Specifies a format string for displaying the value of a property.
- ❖ [RequiredIf]: Specifies that a property is required if another property in the model has a specific value.
- ❖ [CreditCard]: Specifies that a string property must be a valid credit card number format.

- ❖ In ASP.NET MVC, scaffolding is a technique that **generates code** for common web application.
- ❖ The scaffolding code can include controllers, views, and data access code automatically generated that can help developers to create web applications more quickly and with less manual coding.





- ❖ **BUNDLING** - It lets us combine multiple JavaScript (.js) files or multiple cascading style sheet (.css) files, so that they can be downloaded as a unit, rather than making individual HTTP requests for different js and css files.
- ❖ **MINIFICATION** - It squeezes out whitespace and performs other types of compression to make the downloaded files as small as possible.

1. Error handling - Must setup custom error page.
2. Cross-Site-Request-Forgery (CSRF) – Antiforgery token
3. Cross-Site-Scripting (XSS) attacks – Must validate input controls
4. Malicious file upload – Must validate the extension of files.
5. SQL injection attack - Validate inputs, use parameterized queries, use ORM (e.g. dapper , entity framework), use stored procedures and avoid dynamic queries.
6. Save the password in encrypted form so that even developer can't access it.
7. Use https



Your Name :

Your comment :

Upload Jpegs and Gifs only.

Chapter 19 : ASP.NET - Web Forms

Q163. What are the events in Page Life Cycle? V Imp

Q164. What is the difference between `Server.Transfer()` and `Response.Redirect()`?

Q165. What are the different types of Caching?

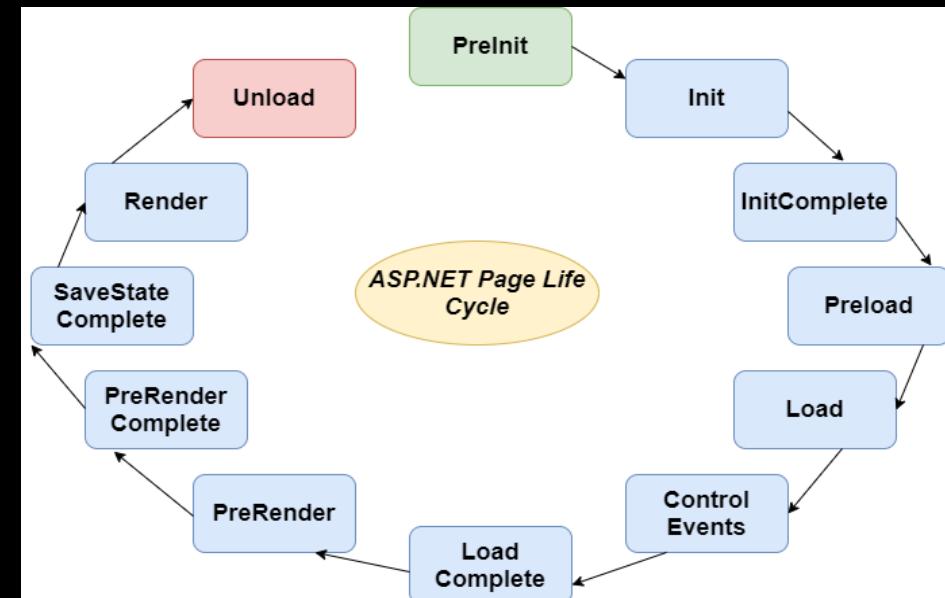
Q166. What are the types of state management? V Imp

Q167. Where the `ViewState` is stored after the page postback?

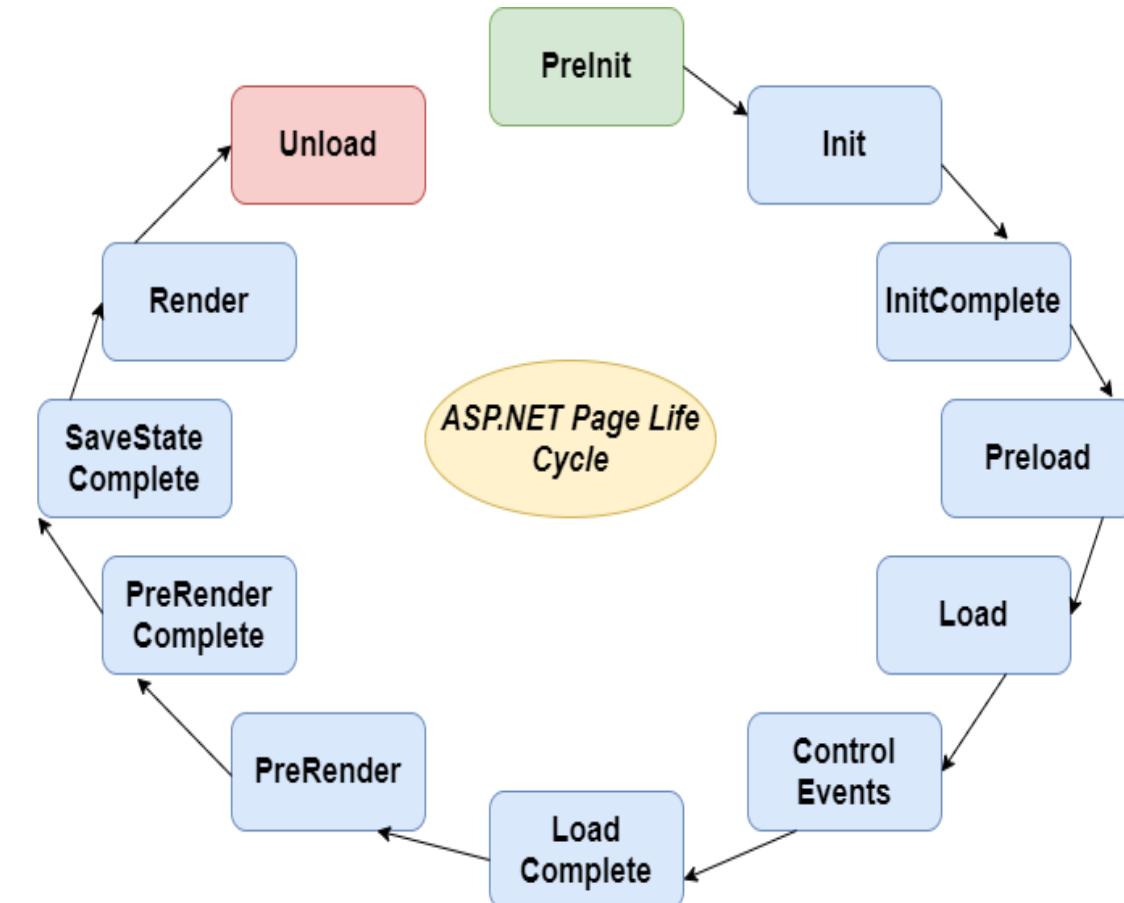
Q168. What are the different ways to store session state in asp.net?

Q169. What is cookie less session?

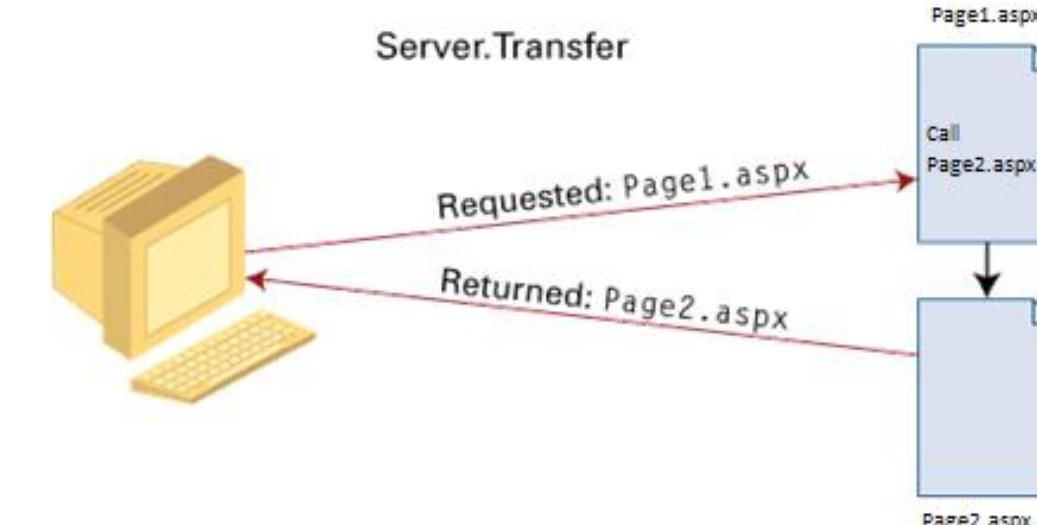
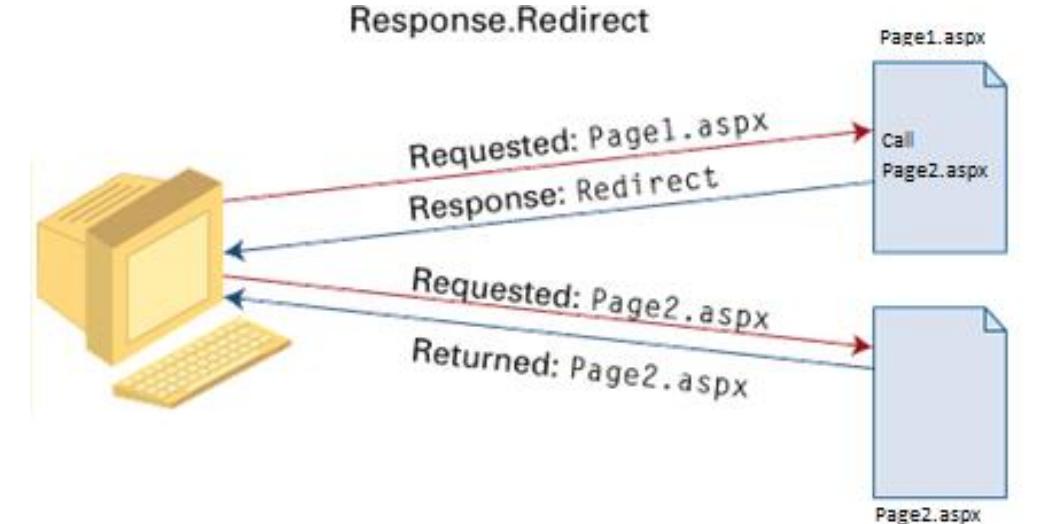
Q170. How to force all the validation controls to run in a page in web forms?

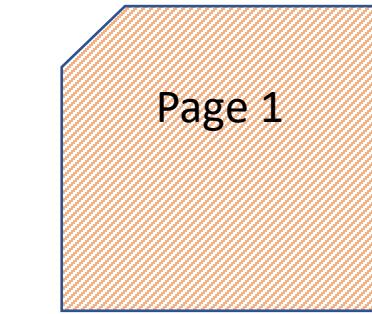


1. **Init** - This event fires after each control has been initialized.
 2. **Preload** – Set properties or default values of the control.
 3. **Load** – All the controls are ready at this event.
 4. **PreRender** - Allows final changes to the page or its control.
 5. **Render** - The Render event generates the client-side HTML
 6. **Unload** - This event is used for cleanup code.
- ❖ In PAGE LOAD event controls are fully loaded.

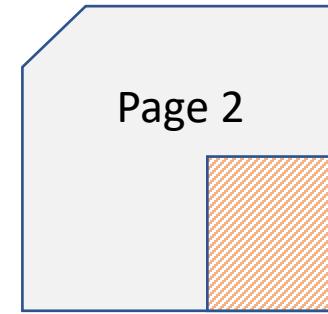


- Both `Response.Redirect()` and `Server.Transfer()` methods are used to **redirect** the user's browser from one page to another page.
- Response.Redirect** uses **round trip** back to the client for redirecting the page.
- It is a **slow** technique, but it maintains the URL history in the client browser for all pages.
- In **Server.Transfer** page processing transfers from one page to the other page without making a round-trip back to the client's browser.
- It is a **faster** technique, but it does not maintain the URL history in the client browser for all pages.

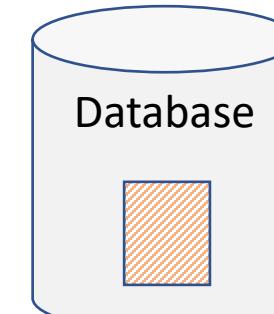




Page Output Caching

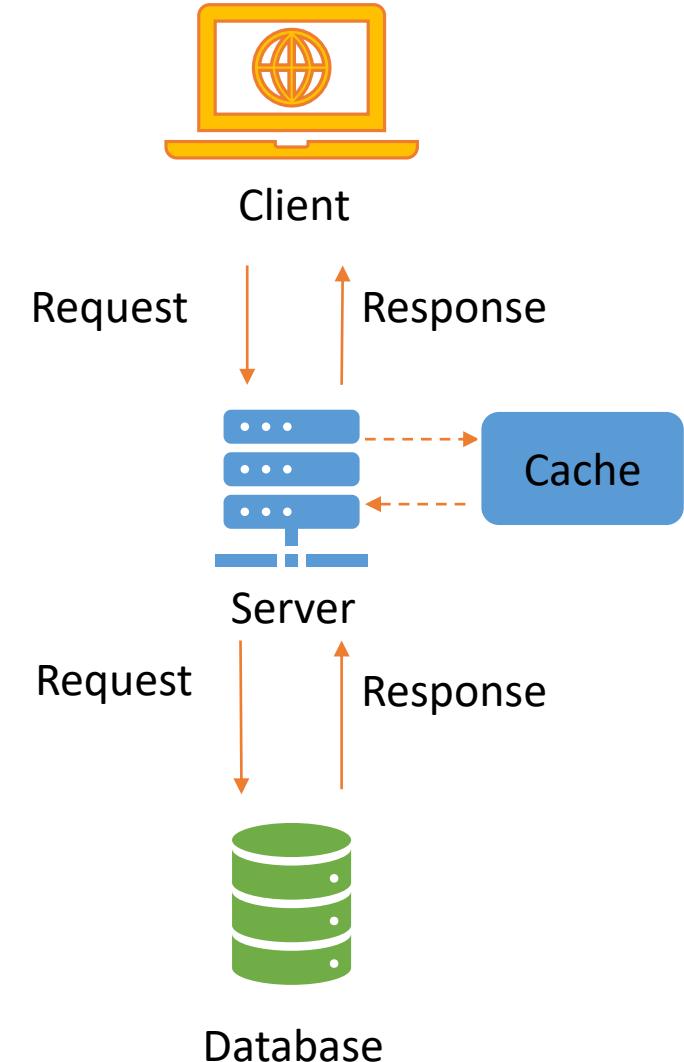


Fragment Caching



Data Caching

1. Page Output Caching: Page output caching allows you to cache the output of an **entire page**. This is useful when a page contains all static content.
2. Fragment Caching: Fragment caching allows you to cache a **portion or fragment** of a page. This is useful when a page contains dynamic content that can be separated from the rest of the page.
3. Data Caching: Data caching allows you to cache **data** retrieved from a database or other data source. This can improve performance by reducing the number of database calls.



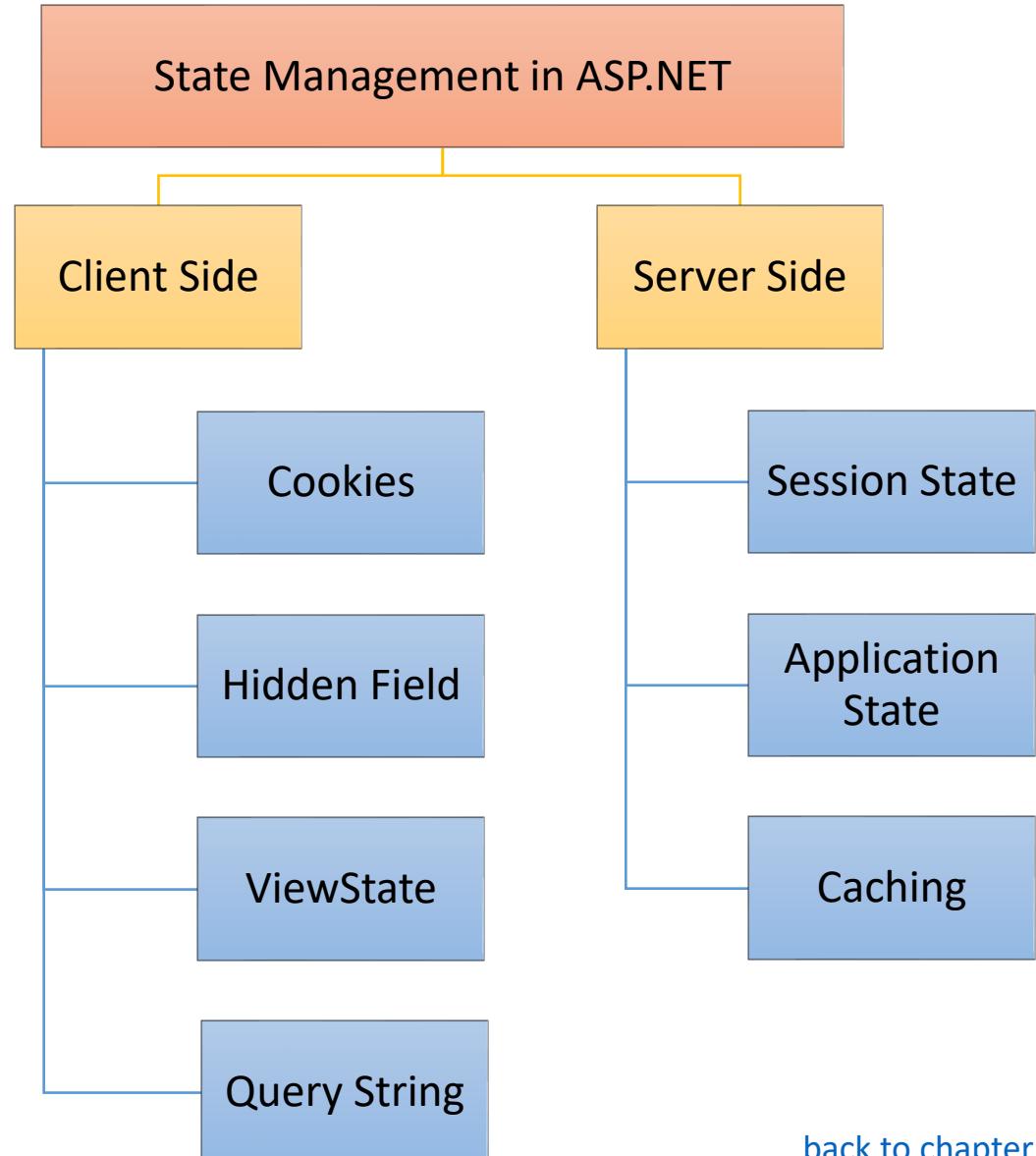
❖ State management refer to **storing** data that is required by a web application to track information between user requests and server responses.

1. Cookies: Cookies is a mechanism for storing small amounts of data on the client's machine. For example, it is used to save usernames.

2. Hidden Fields: Hidden fields are HTML form elements that can be used to store data on the client side. They are often used to pass data between pages.

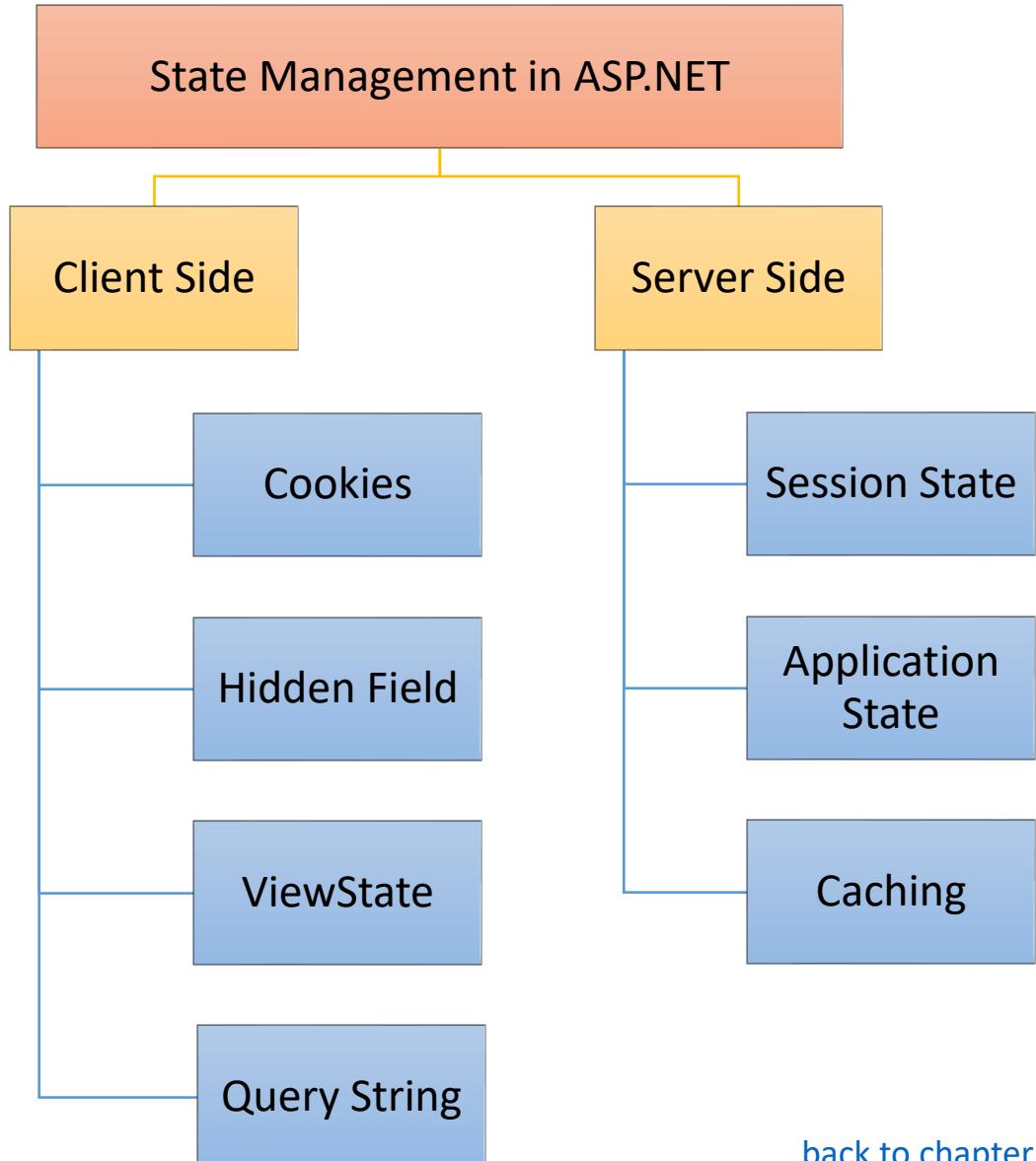
3. ViewState: ViewState is a mechanism that stores the data on the client side as a hidden field. ViewState is useful for persisting data between postbacks in a WebForms application.

4. QueryString: QueryString is a mechanism for passing data from one page to another through the URL. But it is not secure as it is visible to all.



❖ Server-side state management techniques:

1. **Session State:** Session state is a server-side mechanism that allows you to store user-specific data on the server. Session state uses a session ID to associate data with a specific user and can be used to persist data between page requests.
2. **Application State:** Application state is a server-side mechanism that allows you to store data that is shared by all users of an application.
3. **Caching:** Cache is a server-side mechanism that allows you to store frequently accessed data in memory.



- ❖ ViewState is stored in a HIDDEN FIELD on the page at client side.

- ❖ Session state is a mechanism that enables you to store data on server side for multiple requests. It allows you to persist data between pages.

1. In-Process Session State: This is the default session state management mode in ASP.NET, where session data is stored in memory on the same **web server**.
2. State Server Session State: This mode stores session data in a separate process called the ASP.NET State Service, which runs **outside** the web server process.
3. SQL Server Session State: This mode stores session data in a **SQL Server database**, which provides the highest level of reliability and scalability.

Types of storage for session state

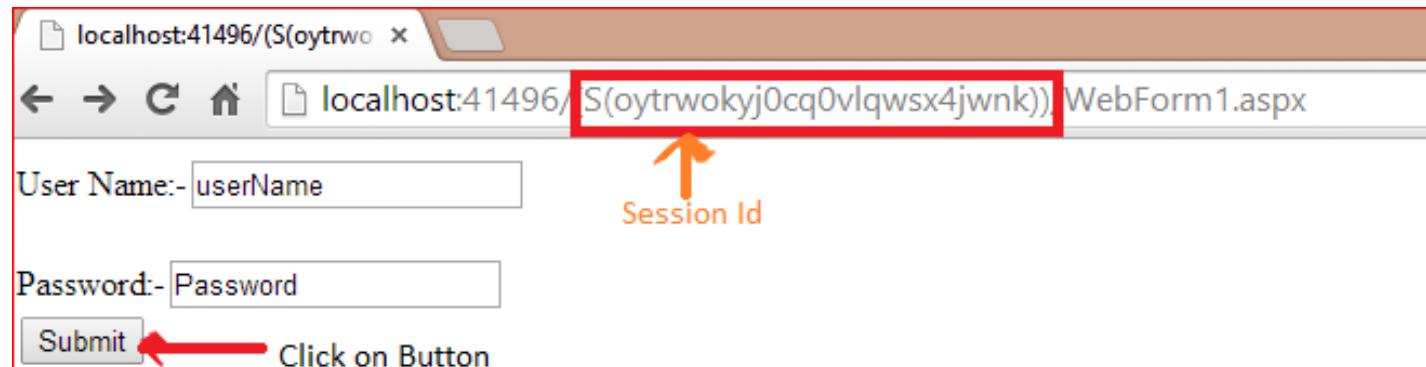


```
<sessionState mode="InProc"/>
```

```
<sessionState mode="StateServer"  
stateConnectionString="tcpip=127.0.0.1:42424"/>
```

```
<sessionState mode="SQLServer"  
sqlConnectionString="data source=myServerAddress;  
initial catalog=myDataBase;user id=myUsername;  
password=myPassword;"/>
```

- ❖ By default, a session uses a browser cookie in the background.
- ❖ In cookie less, the session is passed via **url** instead of cookie.



```
<sessionState mode="InProc" cookieless="true" timeout="20" />
```

- ❖ Page.Validate()

Chapter 20 : ADO.NET & EF

Q171. What are the main **components** of ADO.NET? V Imp

Q172. What is **Connected** architecture and **Disconnected** architecture?

Q173. What are the different **Execute Methods** of ADO.NET?

Q174. What are the **Authentication** techniques used to connect to SQL Server? V Imp

Q175. What is **ORM**? What are the different types of **ORM**?

Q176. What is **Entity Framework**?

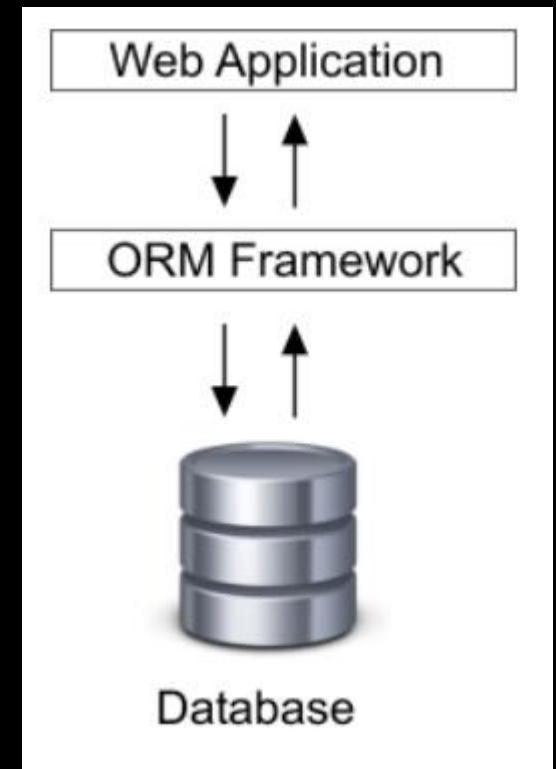
Q177. How will you differentiate ADO.NET from Entity Framework? V Imp

Q178. How **Entity Framework** works? OR How to setup EF?

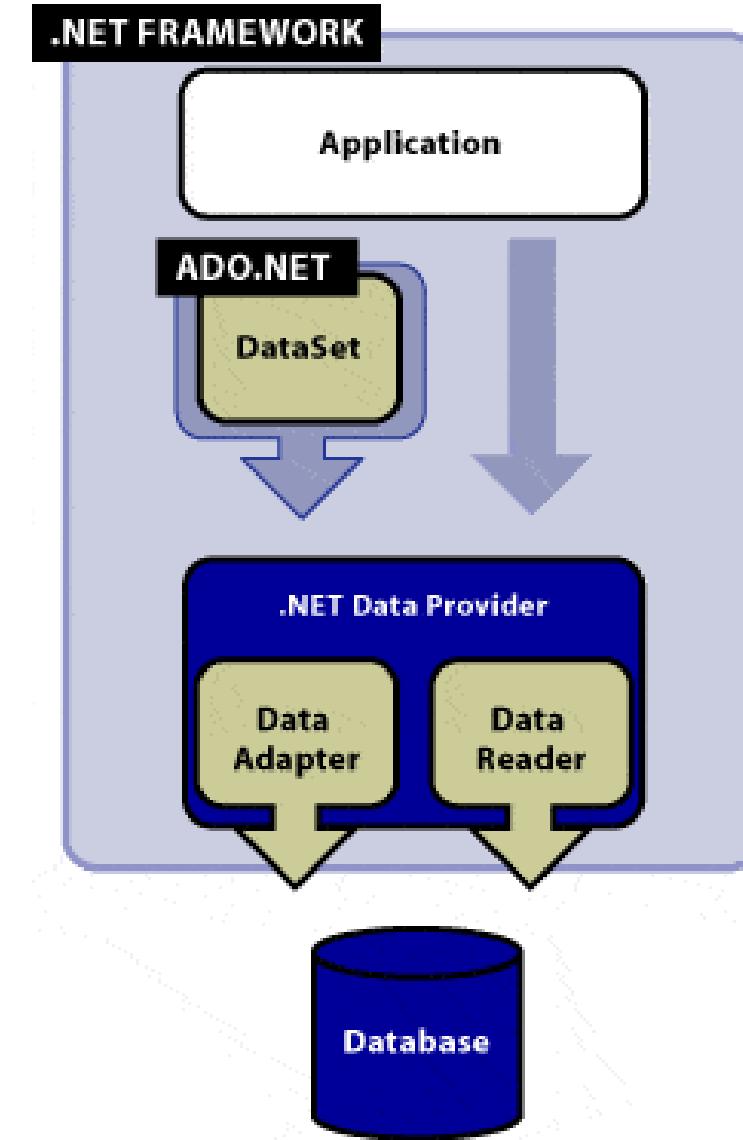
Q179. What is meant by **DbContext** and **DBSet**?

Q180. What are the different **types of application** development approaches used with EF?

Q181. What is the difference between **LINQ to SQL** and **Entity Framework**?



- ❖ **DataSet class** - A DataSet is basically a container which gets the data from one or more tables from the database. It follows disconnected architecture.
- ❖ **DataAdapter class** - A DataAdapter bridges the gap between the disconnected DataSet/ DataTable objects and the physical database.
- ❖ **DataReader Class** - The DataReader allows you to read the data returned by a SELECT command.
 1. It is read only.
 2. Unlike dataset we cannot update the database via this.
 3. It follows connected architecture.

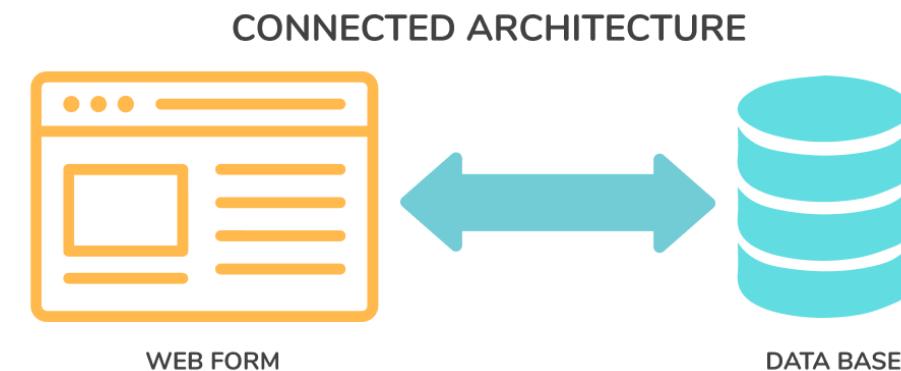
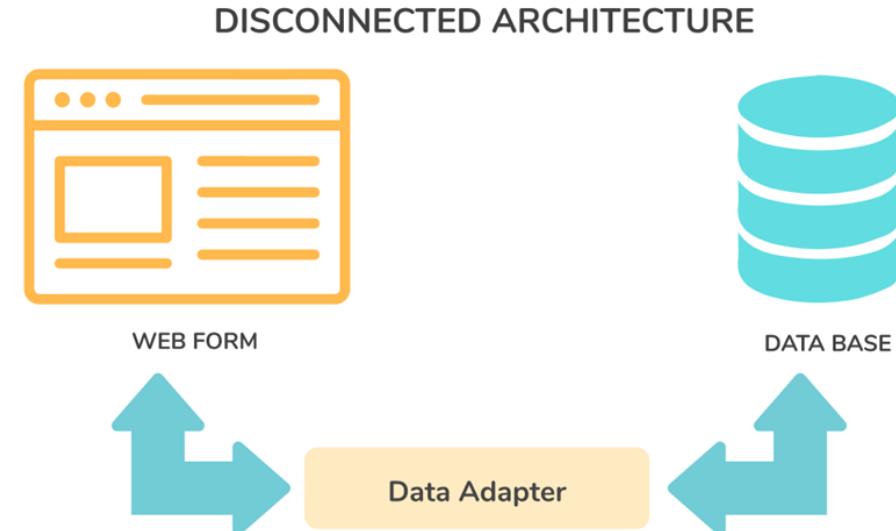


- ❖ **Disconnected Architecture** - Disconnected architecture means, you don't need to connect always to get data from the database.

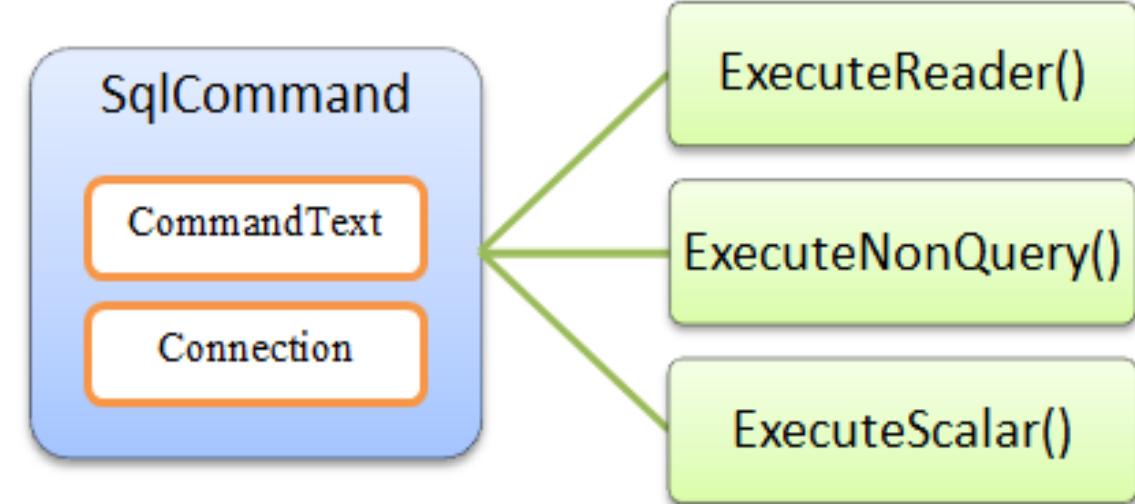
1. Get data into DataAdapter.
2. Manipulate the DataAdapter
3. Resubmit the data.

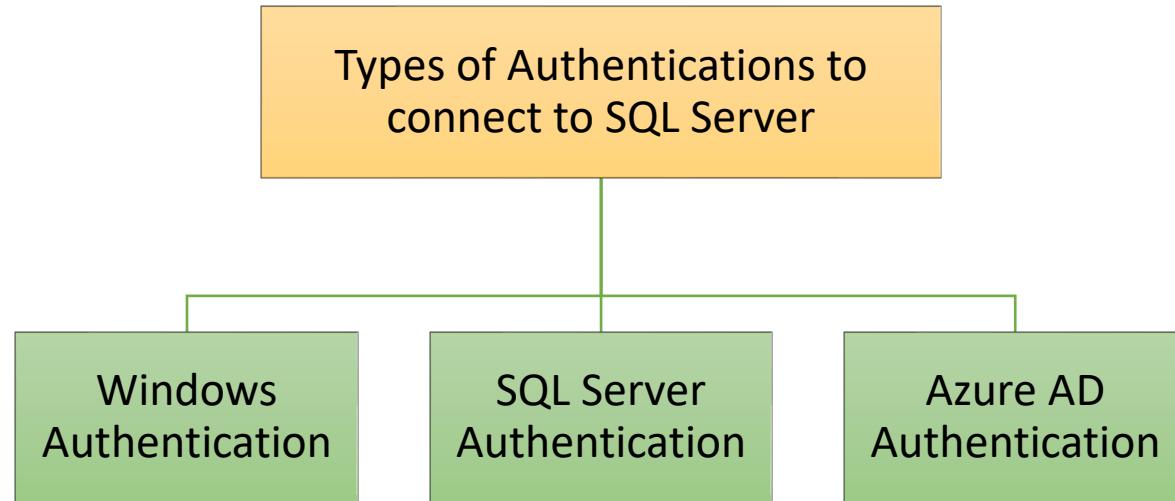
- ❖ It is fast and robust(data will not lose in case of any power failure).

- ❖ **Connected Architecture** - Connected architecture means you are directly interacting with database, but it is less secure and not robust.



1. **EXECUTESCALAR()** - It is used to return SINGLE value from the database.
2. **EXECUTENONQUERY()** – It returns a RESULTSET from database and it has multiple values. It can be used for insert, update and delete.
3. **EXECUTEREADER()** - It only retrieves data from database. No update, insert. It is readonly.





❖ The main authentication techniques are:

1. Windows Authentication: This technique uses the current Windows user account to authenticate to SQL Server. It is the most secure and recommended technique for applications running on a Windows domain.

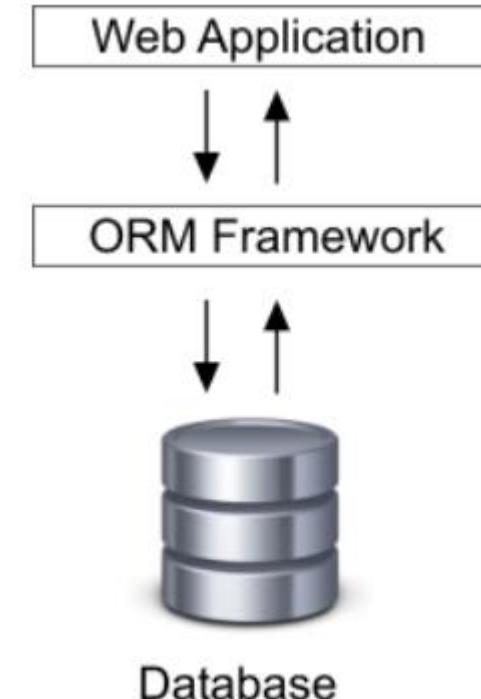
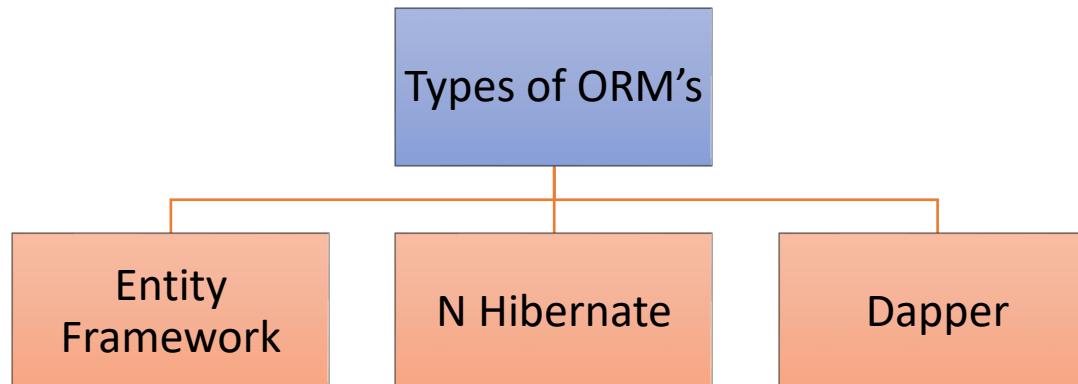
2. SQL Server Authentication: This technique uses a SQL Server user account and password to authenticate to SQL Server.

3. Azure Active Directory Authentication: This technique allows for authentication using Azure Active Directory (AAD) identities.

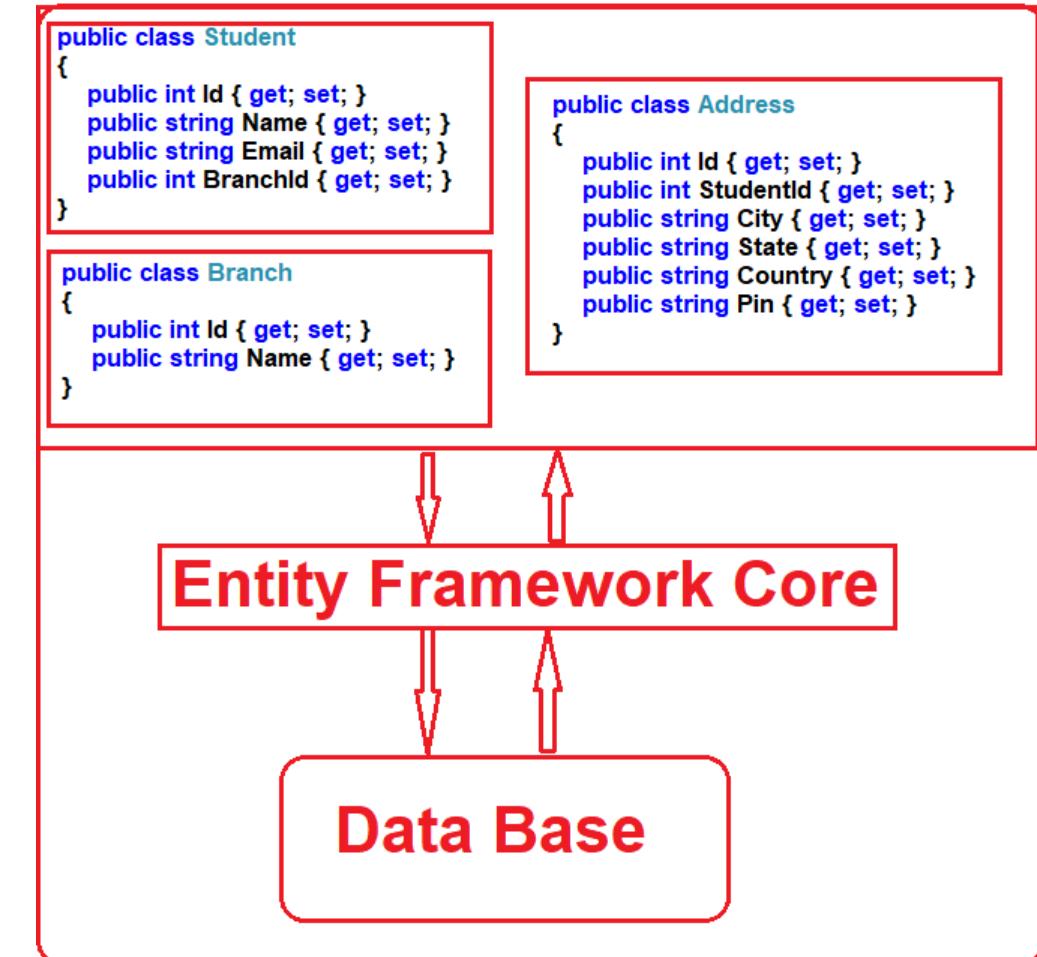
```
<connectionStrings>
  <add name="MyDbConnection"
    connectionString="Data Source=MySqlServer;
    Initial Catalog=MyDatabase;
    Integrated Security=True;" />
</connectionStrings>
```

```
<connectionStrings>
  <add name="MyDbConnection"
    connectionString="Data Source=MySqlServer;
    Initial Catalog=MyDatabase;
    User ID=MyUser;Password=MyPassword;" />
</connectionStrings>
```

- ❖ ORM (Object-Relational Mapper) is for mapping objects in your application with database tables.
- ❖ It is like a wrapper to make database calls simple and easy.



- ❖ An Entity Framework (EF) is an open-source **ORM** (Object-Relational Mapper) from Microsoft.
- ❖ It's like a **wrapper** on ADO.NET.
- ❖ Entity Framework minimizes the coding effort.



ADO.NET

```

public class UserRepository
{
    public DataSet GetAllUsersList()
    {
        DataSet ds = new DataSet(); // Initializes disconnected dataset to fetch results

        //Create and initialize the connection using connection string
        using (SqlConnection sqlConn = SqlConnection ("DataSource=localhost;
                                                Initial Catalog=TestDB; User ID=sa;Password=admin;"))
        {

            sqlConn.Open(); //Open connection

            string sql = "select * from tblusers"; // sql query to access user from table
            SqlCommand sqlCmd = new SqlCommand(sql, sqlConn);

            sqlCmd.CommandType = CommandType.Text; // Define Command Type

            SqlDataAdapter sqlAdapter = new SqlDataAdapter(sqlCmd);

            sqlAdapter.Fill(ds); //Get the data in disconnected mode
        }

        return ds; // return dataset result
    }
}

```

EF

```

public class UserRepository
{
    public static void Main(string[] args)
    {
        // Initializes the dbContext class User Entity
        using (var userContextDB = new UserContext())
        {

            // Create a new user object
            var user = new User() { UserID = "USER-01" };

            // Call Add Command
            userContextDB.User.Add(user);

            // Execute the save command to make changes
            userContextDB.SaveChanges();
        }
    }
}

```

Entity Framework

1. In EF, code is simpler and short.

2. EF can generate SQL statements automatically based on LINQ queries.

3. EF is designed to be database independent, which means same code can work with different databases.

4. EF is slightly slower than ADO.NET because internally EF use ADO.NET only

ADO.NET

In ADO.NET, same code is bigger.

ADO.NET requires developers to write SQL statements themselves.

ADO.NET, is tightly coupled to SQL Server and requires different code for different database systems.

ADO.NET is slightly faster than EF.

❖ How to add records to Student table in database using EF?

1. Install-Package EntityFramework

2. Create the data model for the student entity

3. Create the database context

In the *Models* folder, create a class file named *Student.cs*

```
public class Student
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

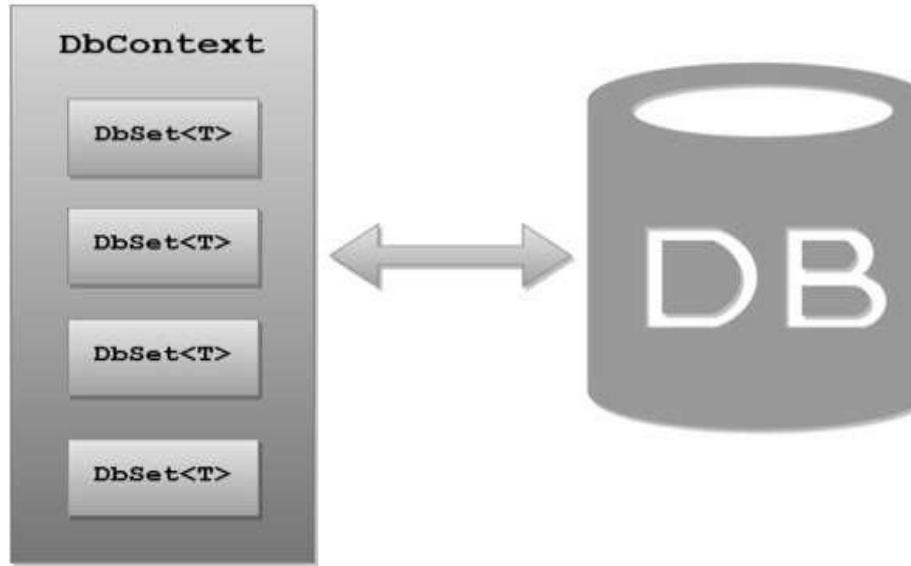
```
public class SchoolContext : DbContext
{
    public DbSet<Student> Students { get; set; }
}
```

3. Add data to student table

```
protected void AddStudents(SchoolContext context)
{
    var students = new List<Student>
    {
        new Student{Name="Happy"},
        new Student{Name="John"},
        new Student{Name="Amit"},
    };

    students.ForEach(s => context.Students.Add(s));

    context.SaveChanges()
}
```

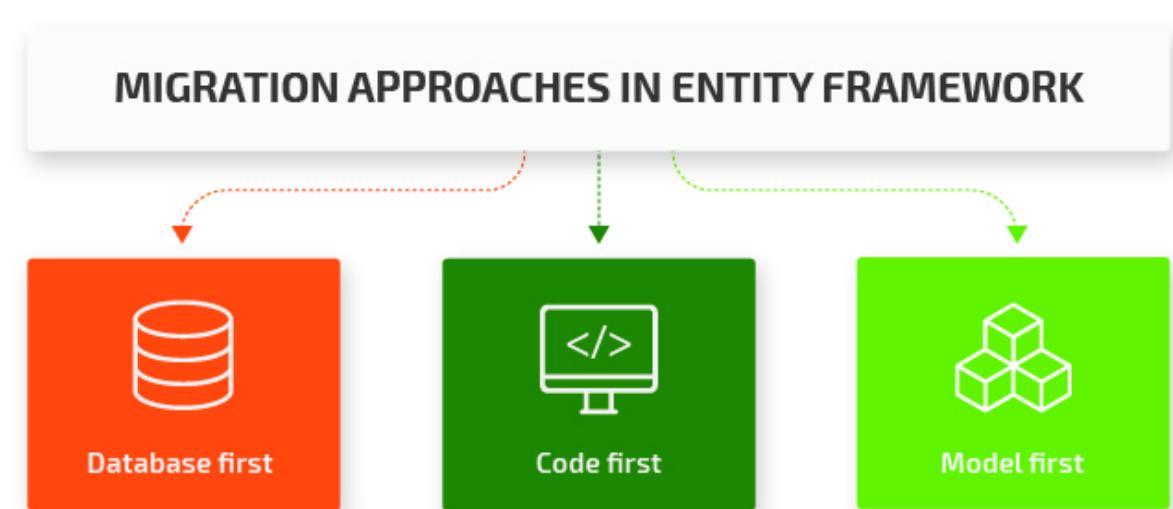


- ❖ DbContext is a class in the Entity Framework that helps in creating the **communication** between the database and the domain/entity class.
- ❖ The DbSet class represents an **entity set** that can be used for create, read, update, and delete operations.

```
public class MyDbContext : DbContext
{
    public MyDbContext() : base("MyDbConnectionString")
    {
        // optional: disable database initialization
        Database.SetInitializer<MyDbContext>(null);
    }

    public DbSet<Customer> Customers { get; set; } // Db
    // add more DbSet properties for other tables here
}
```

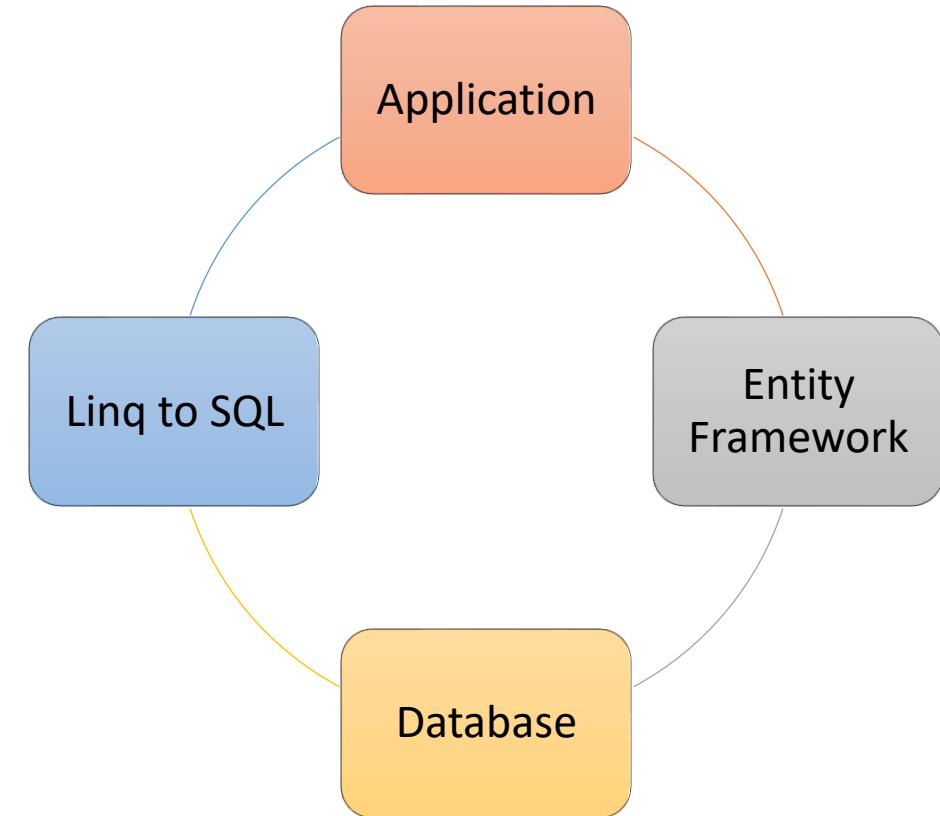
- 1. Database First** – In Database First approach first the database is created and then the entity model is generated from it.
- 2. Code First** - The Code First approach involves creating the data model using C# or VB.NET classes. Developers can define entities, relationships, and other schema elements using code, and then generate the database schema from the code.
- 3. Model First** - The Model First approach involves creating the data model using visual tools such as Entity Data Model Designer in Visual Studio. Developers can create entities, relationships, and other schema elements visually, and then generate the database schema from the model.



- ❖ LINQ to SQL and Entity Framework are both Object-Relational Mapping (ORM) frameworks that allow developers to interact with relational databases using object-oriented programming techniques.

- ❖ Differences between LINQ to SQL and EF:
 1. LINQ to SQL is a lightweight ORM that supports only SQL Server databases.
 2. Entity Framework is a more robust ORM that supports multiple database providers, including SQL Server, Oracle, MySQL etc.

 3. LINQ to SQL is suitable for small to medium-sized applications.
 4. Entity Framework is suitable for large and complex applications.



Chapter 21 : Web API - Basics

Q182. What is Web API? What is the purpose of Web API?

Q183. What are Web API advantages over WCF and web services? V Imp

Q184. What are HTTP verbs or HTTP methods?

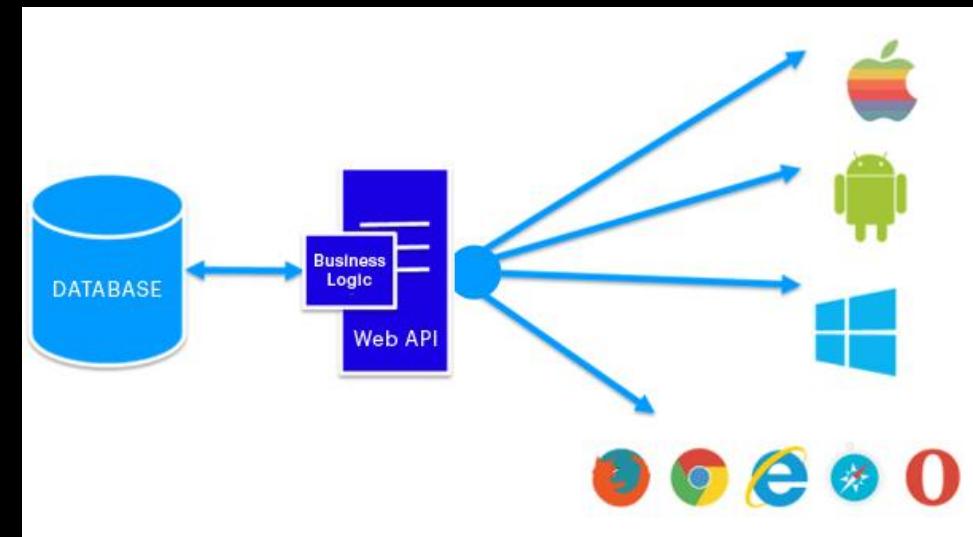
Q185. What is the difference Rest API and Web API?

Q186. What are REST guidelines? What is the difference between Rest and Restful?

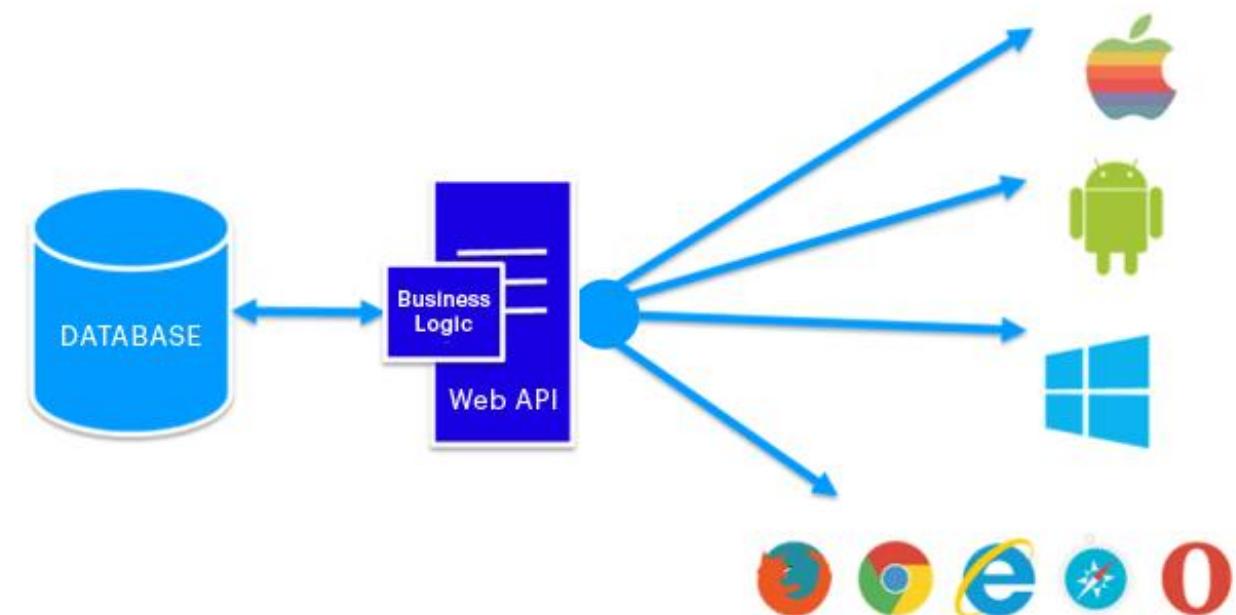
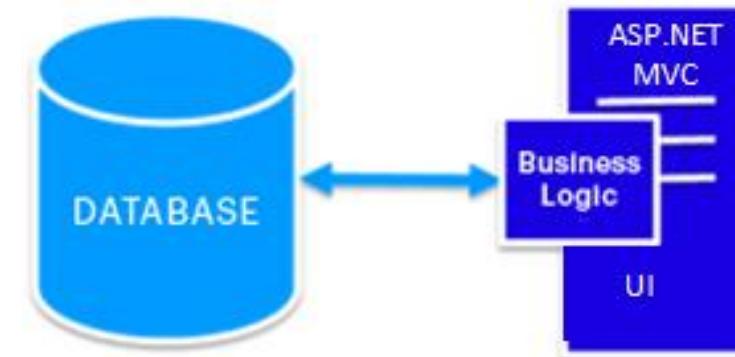
Q187. Is it possible to use WCF as Restful services?

Q188. How to consume Web API from a .NET MVC application?

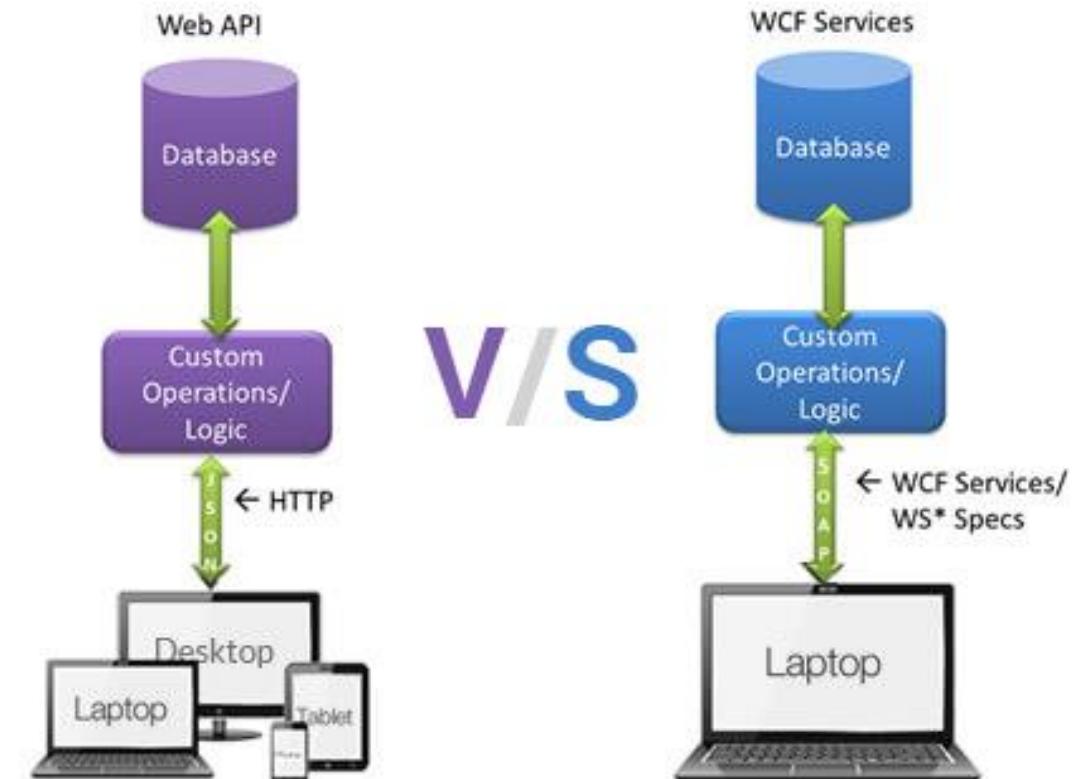
Q189. What is the difference between Web API and MVC Controller?

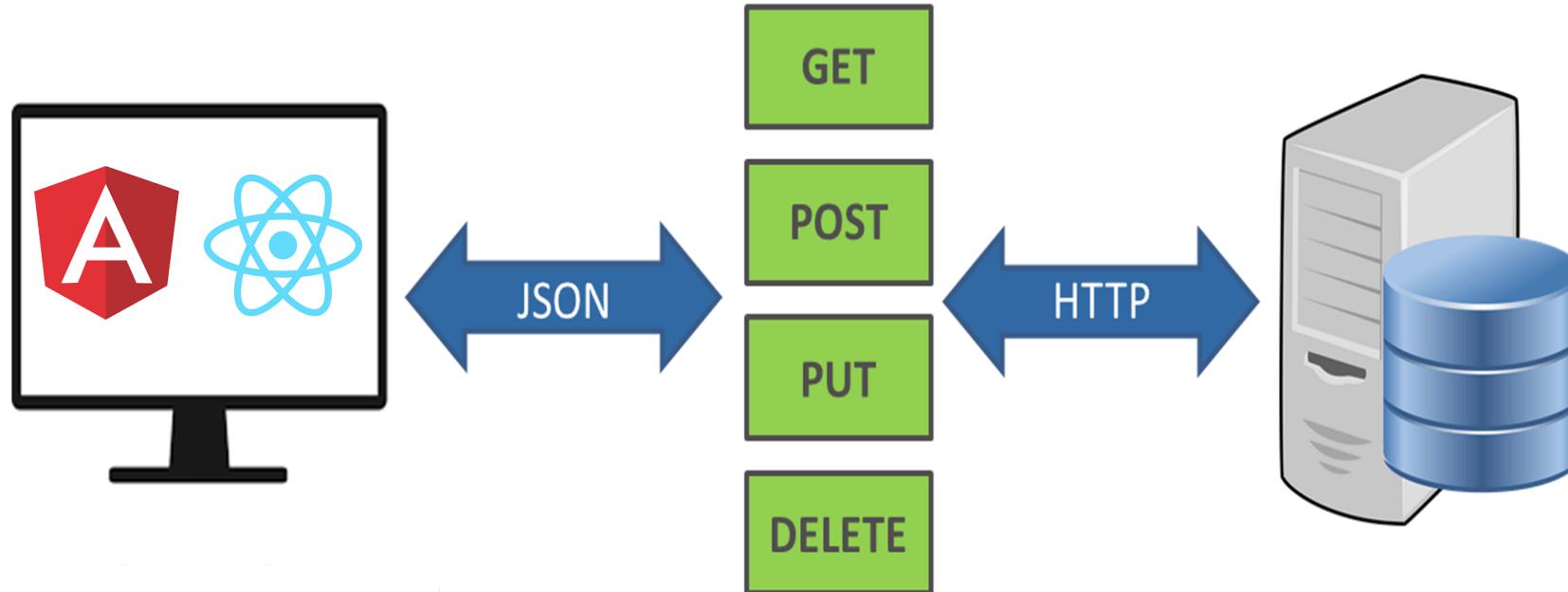


- ❖ Web API(Application Programming Interface) are HTTP based services that can be accessed from browser or mobile-apps.
- ❖ Web API enables different software applications to communicate with each other through the internet.
- ❖ Web API's mostly use JSON and XML formats to transfer data between different applications.

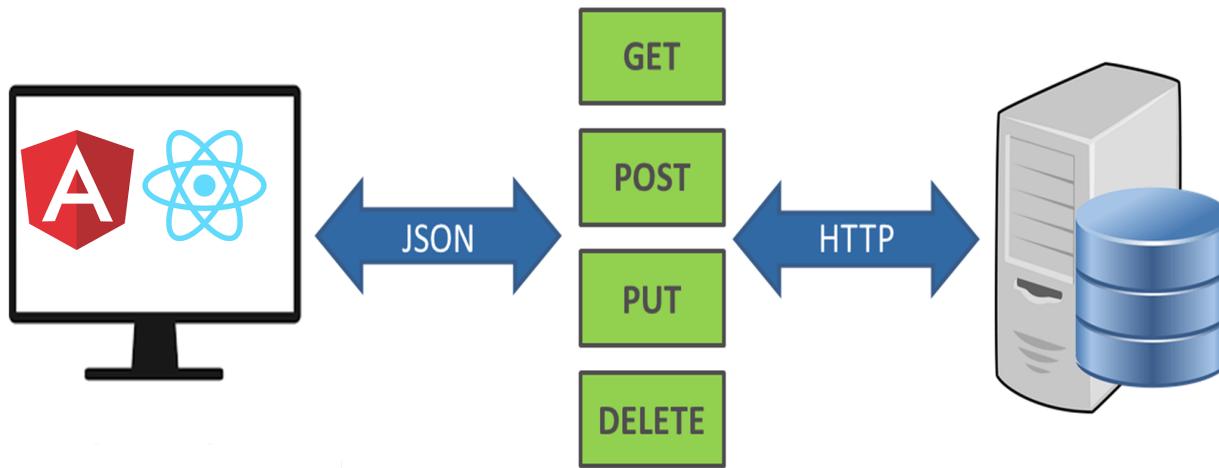


1. Web APIs are **simpler** to use and learn, and use familiar web technologies like HTTP, JSON and XML.
2. Web APIs are more **flexible**, can be hosted on a wide range of platforms, and are compatible with many devices including mobile devices, web browsers, and IoT devices.
3. Web APIs are more **scalable** and can handle a large number of requests without consuming too many resources.
4. Web APIs are **faster** than WCF because they use lightweight protocols like JSON and HTTP.





HTTP Method	Action	Examples
GET	Obtain information about a resource	<code>http://example.com/api/orders</code> (retrieve order list)
GET	Obtain information about a resource	<code>http://example.com/api/orders/123</code> (retrieve order #123)
POST	Create a new resource	<code>http://example.com/api/orders</code> (create a new order, from data provided with the request)
PUT	Update a resource	<code>http://example.com/api/orders/123</code> (update order #123, from data provided with the request)
DELETE	Delete a resource	<code>http://example.com/api/orders/123</code> (delete order #123)



Screenshot of a Fiddler or similar network traffic capture tool interface:

Parsed (selected tab), **Raw**, **Scratchpad**, **Options**

Method: POST **Request URL:** `http://localhost:60464/api/student?age=25`

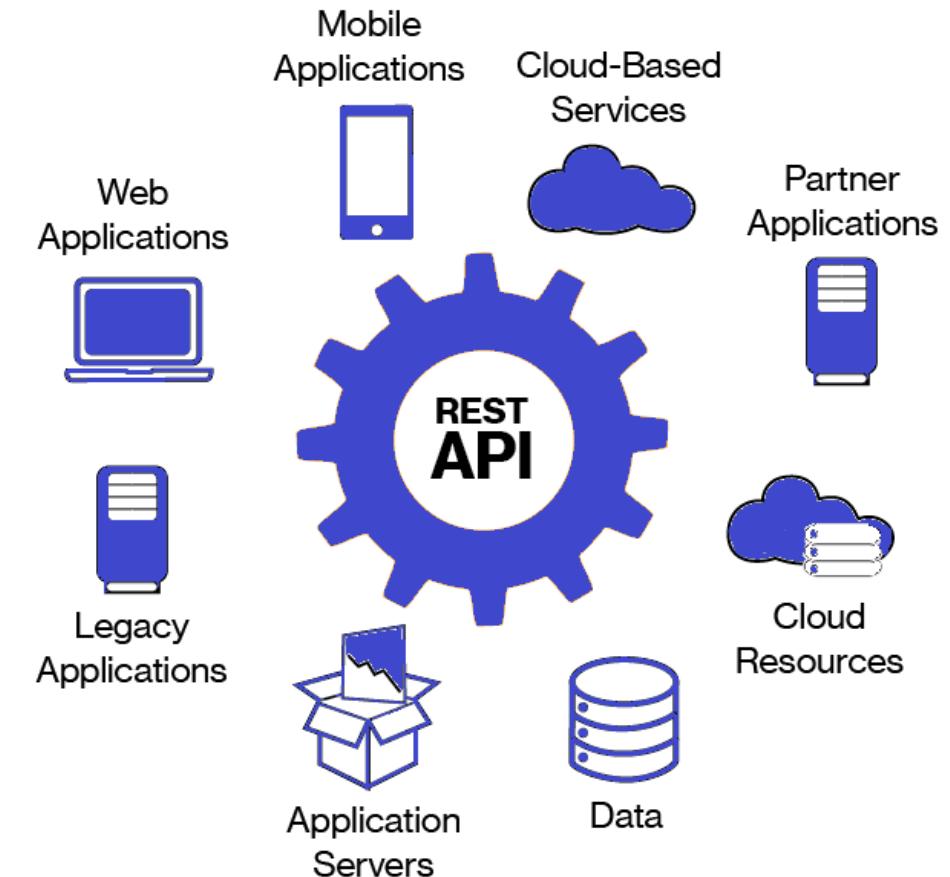
Headers:

- User-Agent: Fiddler
- Host: localhost:60464
- Content-Type: application/json
- Content-Length: 31

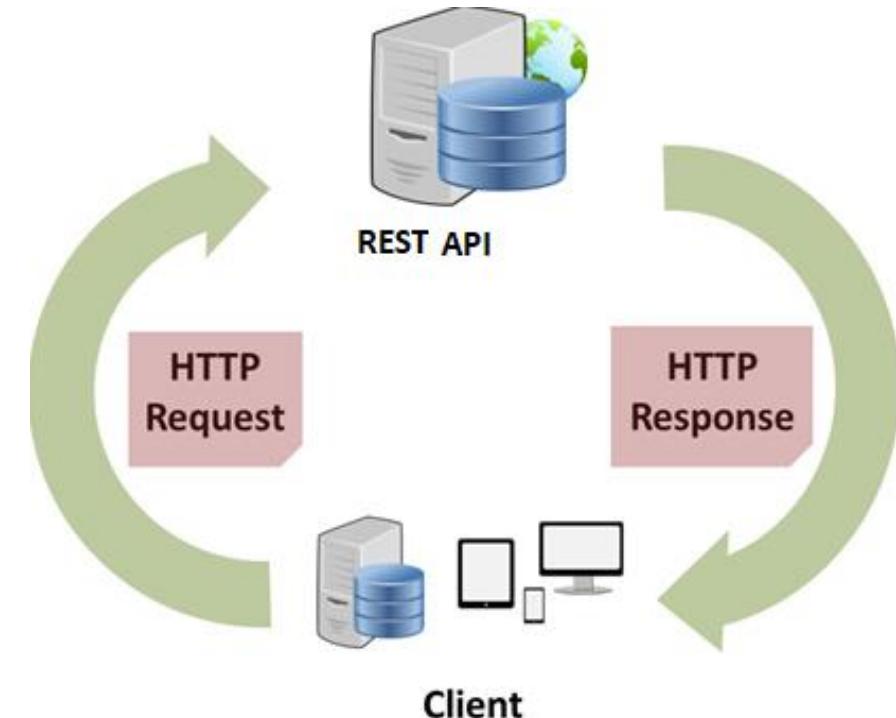
Request Body:

```
{  
  id:1,  
  name:'steve'  
}
```

- ❖ REST is a **set of guidelines** which helps in creating a system where applications can easily communicate with each other.
- ❖ REST stands for Representational State Transfer.



1. **Separation of Client and Server** - The implementation of the client and the server must be done independently.
 2. **Stateless** - The server will not store anything about the latest HTTP request the client made. It will treat every request as new request.
 3. **Uniform interface** – Identify the resources by URL
www.abc.com/api/questions
 4. **Cacheable** – The API response should be cacheable to improve the performance.
 5. **Layered system** - The system should follow layered pattern.
For example, MVC is a layered system.
- ❖ **RESTFUL Service:** If a system written by applying REST guidelines, then it is also called REST service or RESTful services.



- ❖ Yes, we can develop RESTful services with WCF by following the REST guidelines in WCF service.

- ❖ Web API methods can be consumed with the help of **HttpClient** class.

2. Set the `BaseAddress` property of client object.

4. `GetAsync` method will send request to find api resource `GetAllEmployees`. And then store the response in `HttpResponseMessage` class object.

6. If successful then result is saved in a variable

1. Create the object of `HttpClient` class.

3. Add media type, the request format of data. Here it is json.

5. Checking the response is successful or not.

7. Then the variable will be deserialized from json format to `Employee` object.

```
public class HomeController : Controller
{
    string Baseurl = "http://facebook.com/api";

    public async Task<ActionResult> Index()
    {
        List<Employee> EmpInfo = new List<Employee>();

        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri(Baseurl);
            client.DefaultRequestHeaders.Clear();

            client.DefaultRequestHeaders.Accept.Add(new
                MediaTypeWithQualityHeaderValue("application/json"));

            HttpResponseMessage Res = await client.GetAsync("Employee/GetAllEmployees");

            if (Res.IsSuccessStatusCode)
            {
                var EmpResponse = Res.Content.ReadAsStringAsync().Result;
                EmpInfo = JsonConvert.DeserializeObject<List<Employee>>(EmpResponse);
            }
            return View(EmpInfo);
        }
    }
}
```

[back to chapter index](#)

Web API Controller	MVC Controller
1. Web api controller derives from SYSTEM.WEB.HTTP.APICO NTROLLER class.	ASP.NET MVC controller derives from SYSTEM.WEB.MVC.CON TROLLER class.
2. Web API controller does not give view support.	ASP.NET MVC gives view support.

❖ Web API Controller

```
[ApiController]  
[Route("[controller]")]  
public class WeatherForecastController : ControllerBase  
{  
    ...  
}
```

```
[Route("[controller]")]  
public class WeatherForecastController : ApiController  
{  
    ...  
}
```

❖ MVC Controller

```
public class HomeController : Controller  
{  
    ...  
}
```

```
...  
public abstract class Controller : ControllerBase, IActionFilter, IFilterMetadata,  
{  
    ...  
}
```

Chapter 22 : Web API – Authentication & JWT

Q190. What are the **types of authentication** techniques in web api?

Q191. What is **Basic Authentication** in Web API?

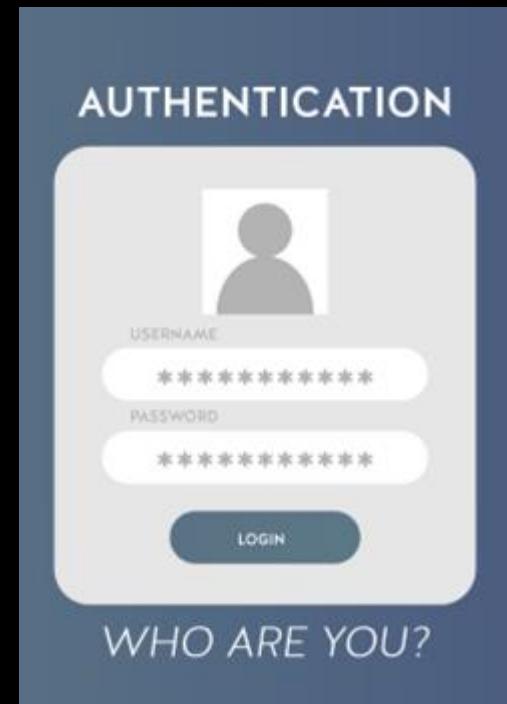
Q192. What is **API Key Authentication** in Web API?

Q193. What is **Token** based authentication? **V Imp**

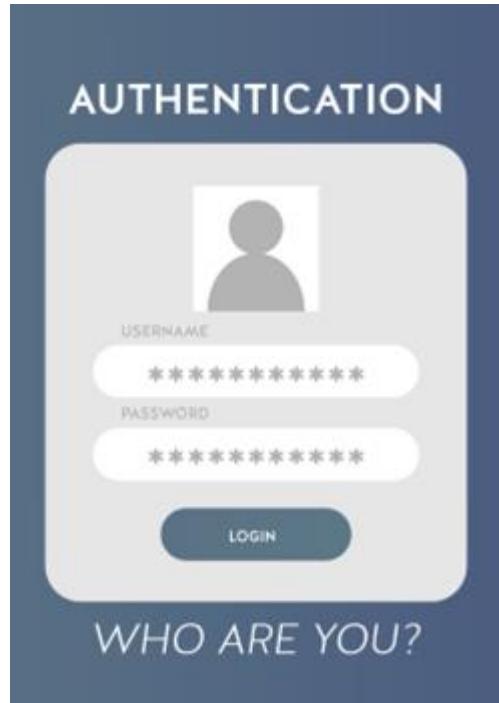
Q194. What is **JWT** Authentication? **V Imp**

Q195. What are the **parts of JWT** token?

Q196. Where JWT token reside in the request?



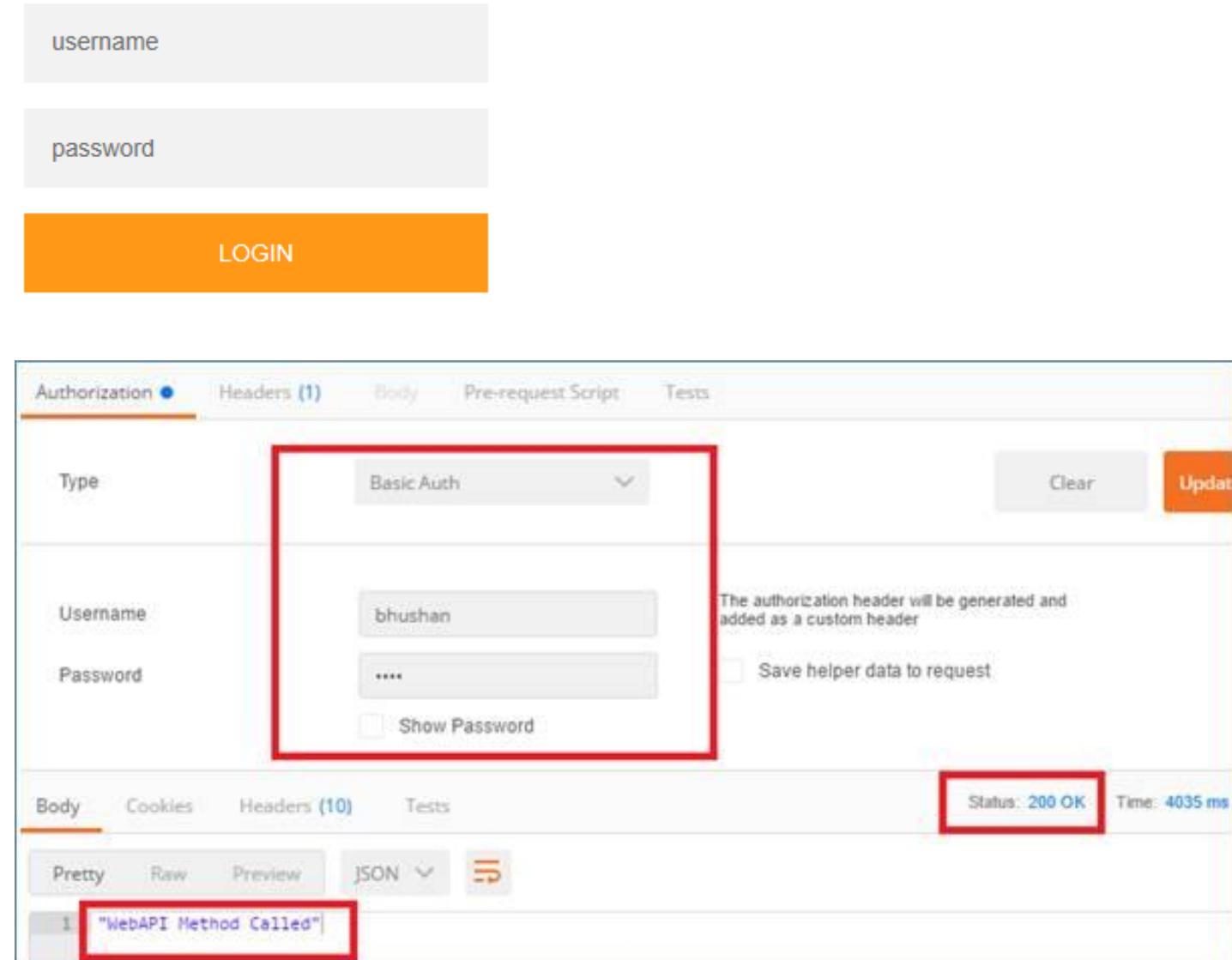
- ❖ Authentication is the process of verifying the identity of a user by validating their credentials such as username and password.



Types of Authentication techniques

- Basic Authentication
- API Key Authentication
- Token-based Authentication
- Certificate-based Authentication
- Windows Authentication
- Multi-Factor Authentication

- ❖ In Basic Authentication, the user passes their credentials on a post request. At the WebAPI end, credentials are verified, and response is sent back.
- ❖ The disadvantage of it is, Basic Authentication sends credentials in plain text over the network, so it is not considered a secure method of authentication.



The screenshot shows a Postman request configuration and its response. The request is set to 'Basic Auth' with 'bhushan' as the username and a masked password. The response status is '200 OK' with a response body containing 'WebAPI Method Called'.

username

password

LOGIN

Authorization Headers (1) Body Pre-request Script Tests

Type: Basic Auth

Username: bhushan

Password: Show Password

The authorization header will be generated and added as a custom header Save helper data to request

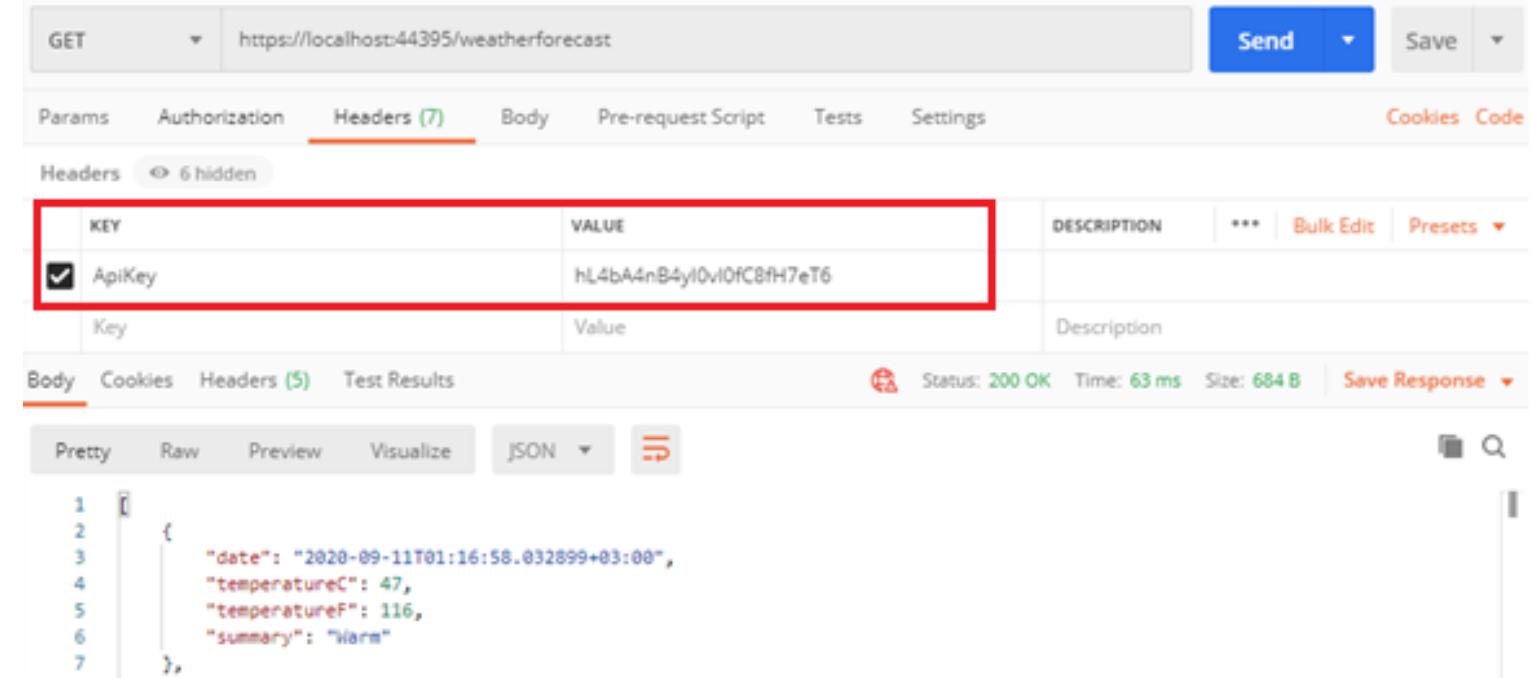
Status: 200 OK Time: 4035 ms

Body Cookies Headers (10) Tests

Pretty Raw Preview JSON

1 "WebAPI Method Called"

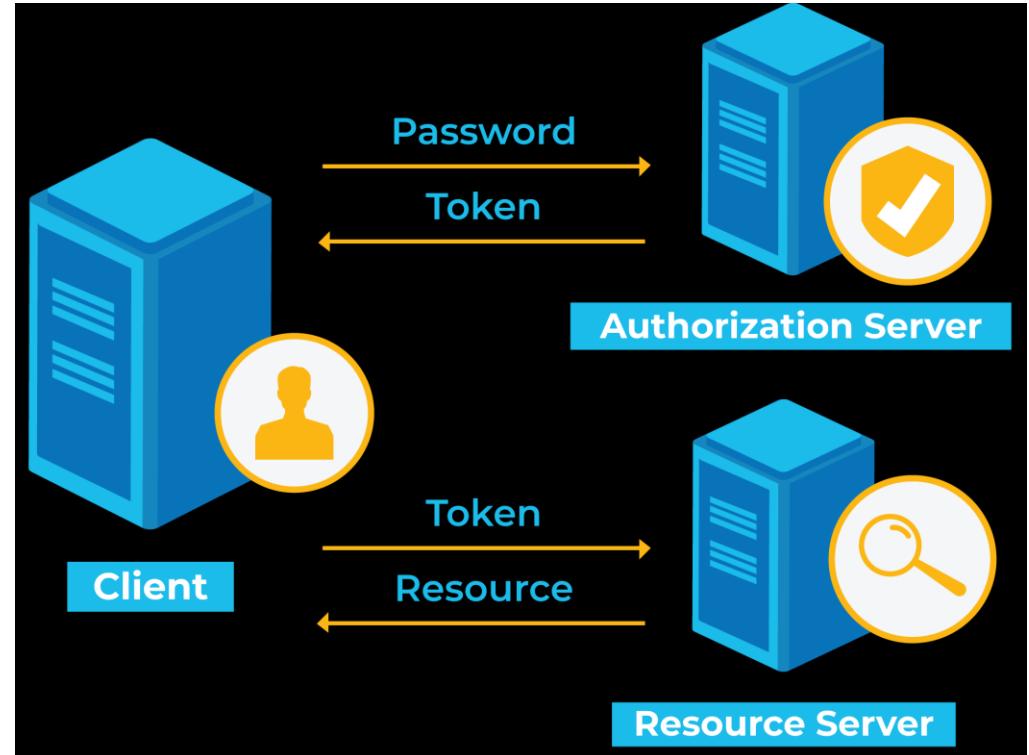
- ❖ **API Key Authentication** - In API Key Authentication, the API owner will share an API key with the users and this key will authenticate the users of that API.
- ❖ The disadvantage of it is, API keys can be shared or stolen therefore it may not be suitable for all scenarios.



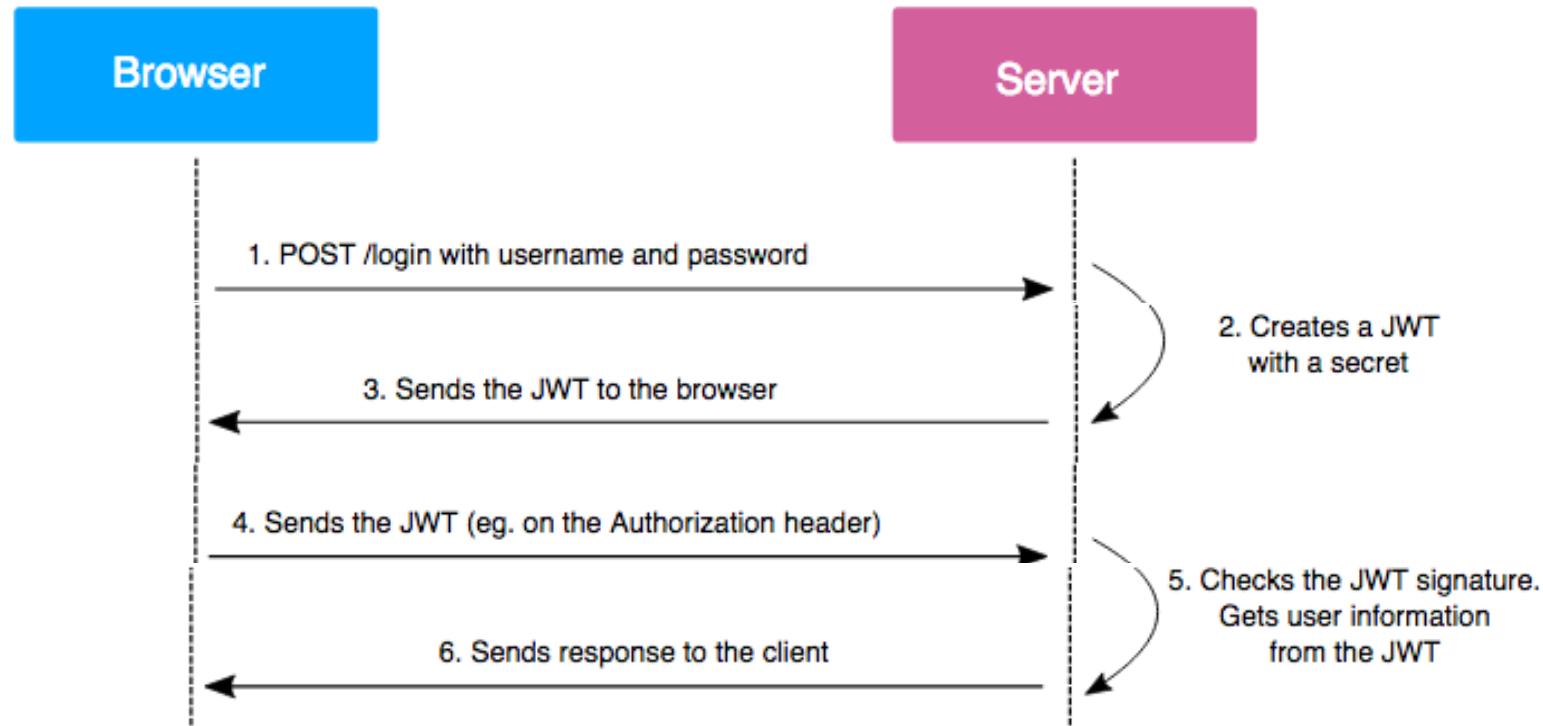
The screenshot shows the Postman application interface. At the top, it displays a GET request to `https://localhost:44395/weatherforecast`. The 'Headers' tab is selected, showing a table with one row: `ApiKey` (Key) and `hL4bA4nB4yI0vI0fC8fH7eT6` (Value). This row is highlighted with a red box. Below the table, the 'Body' tab is selected, showing a JSON response:

```
1 [  
2   {  
3     "date": "2020-09-11T01:16:58.032899+03:00",  
4     "temperatureC": 47,  
5     "temperatureF": 116,  
6     "summary": "Warm"  
7   },  
8 ]
```

- ❖ Token-based authentication is a 4 step process:
 1. Client application first sends a request to Authentication server with valid **credentials**.
 2. The Authentication server/ Web API sends an **Access token** to the client as a response.
 3. In next request, the client uses the same token to access the restricted resources until the token is **valid** or not expired.
 4. If the Access token is expired, then the client application can request for a new access token by using **Refresh token**.



- ❖ JWT authentication is a token base authentication where JWT is a token format.
- ❖ JWT stands for JSON Web Token.



- ❖ JWT token has 3 parts:
 1. Header
 2. Payload
 3. Signature

JWT Token

Encoded

Decoded

HEADER:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT:

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
) secret base64 encoded
```

Signature Verified

Algorithm used to generate the token.

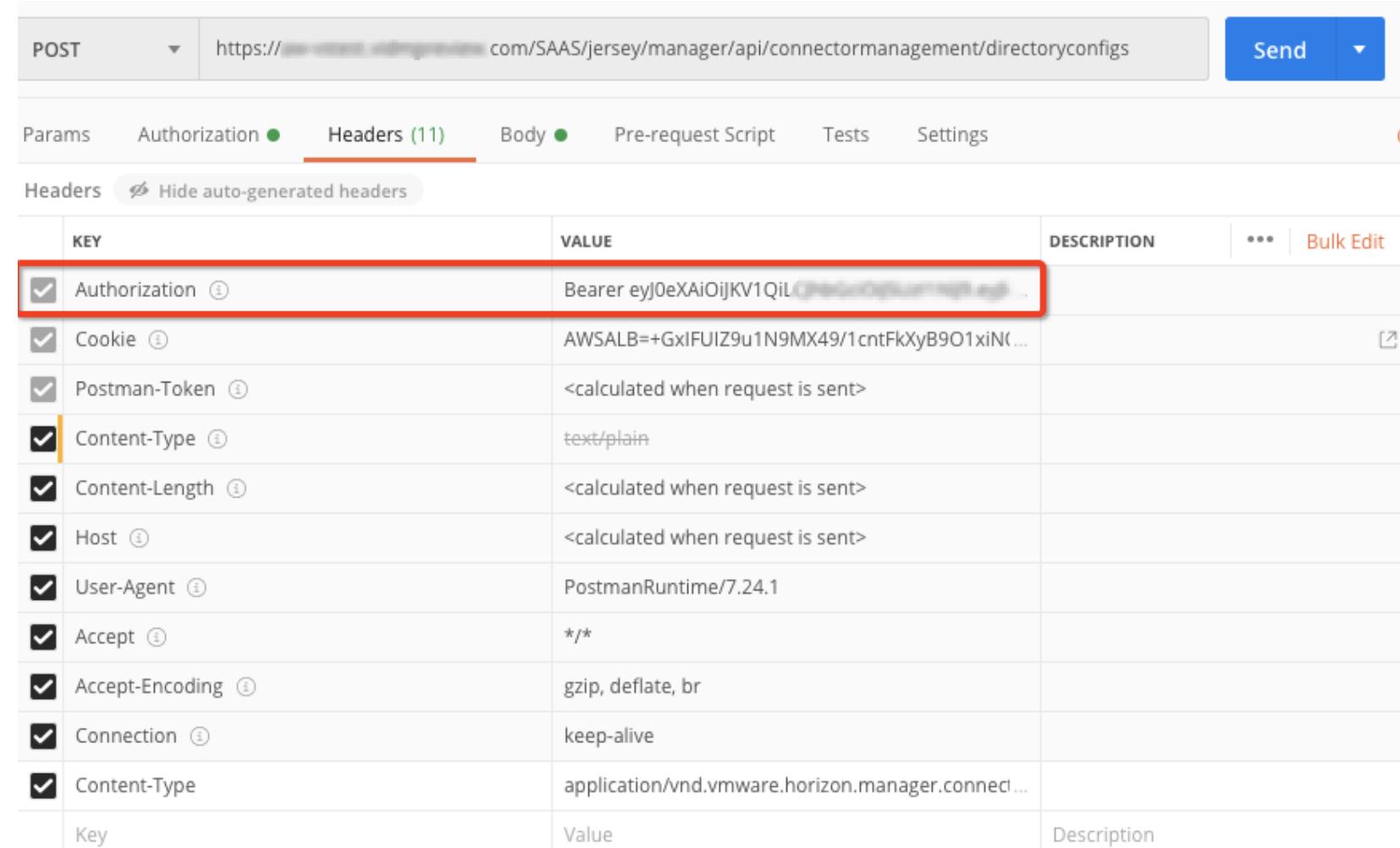
The type of the token, which is JWT here.

The payload contains the CLAIMS.

The signature is a string that is used to ensure the INTEGRITY of the token and verify that it has not been tampered with.

❖ In REQUEST HEADER.

KEY - Authorization
VALUE - Token



The screenshot shows the Postman interface with a 'POST' request to 'https://[REDACTED].com/SAAS/jersey/manager/api/connectormanagement/directoryconfigs'. The 'Headers' tab is selected, displaying 11 headers. The 'Authorization' header is highlighted with a red box, showing its value as 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8v[REDACTED]'. Other headers listed include 'Cookie', 'Postman-Token', 'Content-Type', 'Content-Length', 'Host', 'User-Agent', 'Accept', 'Accept-Encoding', 'Connection', and 'Content-Type' (again). The 'Body' tab is also visible.

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8v[REDACTED]	
Cookie	AWSALB=+GxIFUIZ9u1N9MX49/1cntFkXyB9O1xiN(...	
Postman-Token	<calculated when request is sent>	
Content-Type	text/plain	
Content-Length	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.24.1	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Content-Type	application/vnd.vmware.horizon.manager.connect	

Chapter 23 : Web API - More

Q197. How to test Web API? What are the tools?

Q198. What are main **Return Types** supported in Web API? **V Imp**

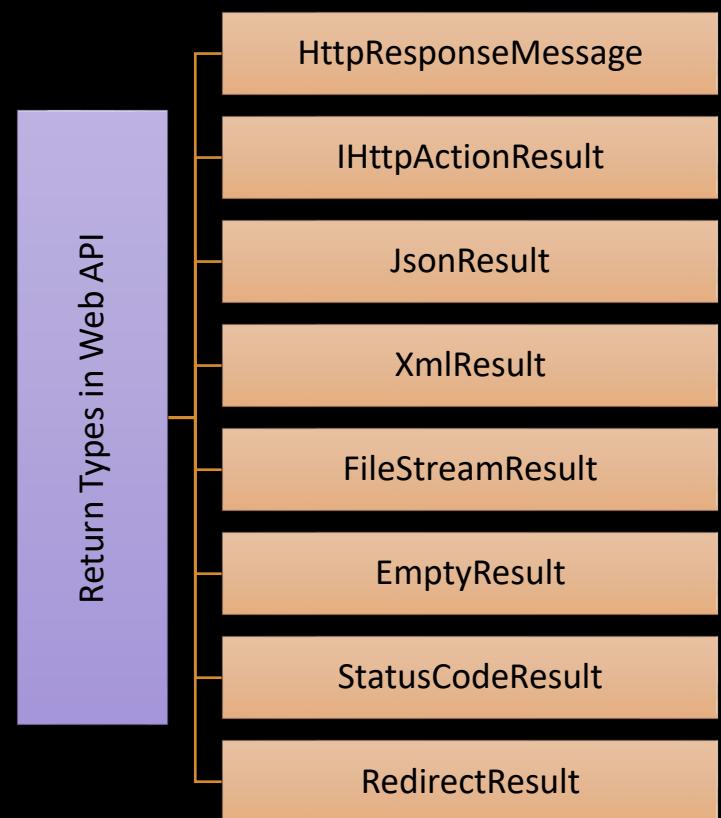
Q199. What is the difference between **HttpResponseMessage** and **IHttpActionResult**?

Q200. What is the difference between **IActionResult** and **IHttpActionResult**?

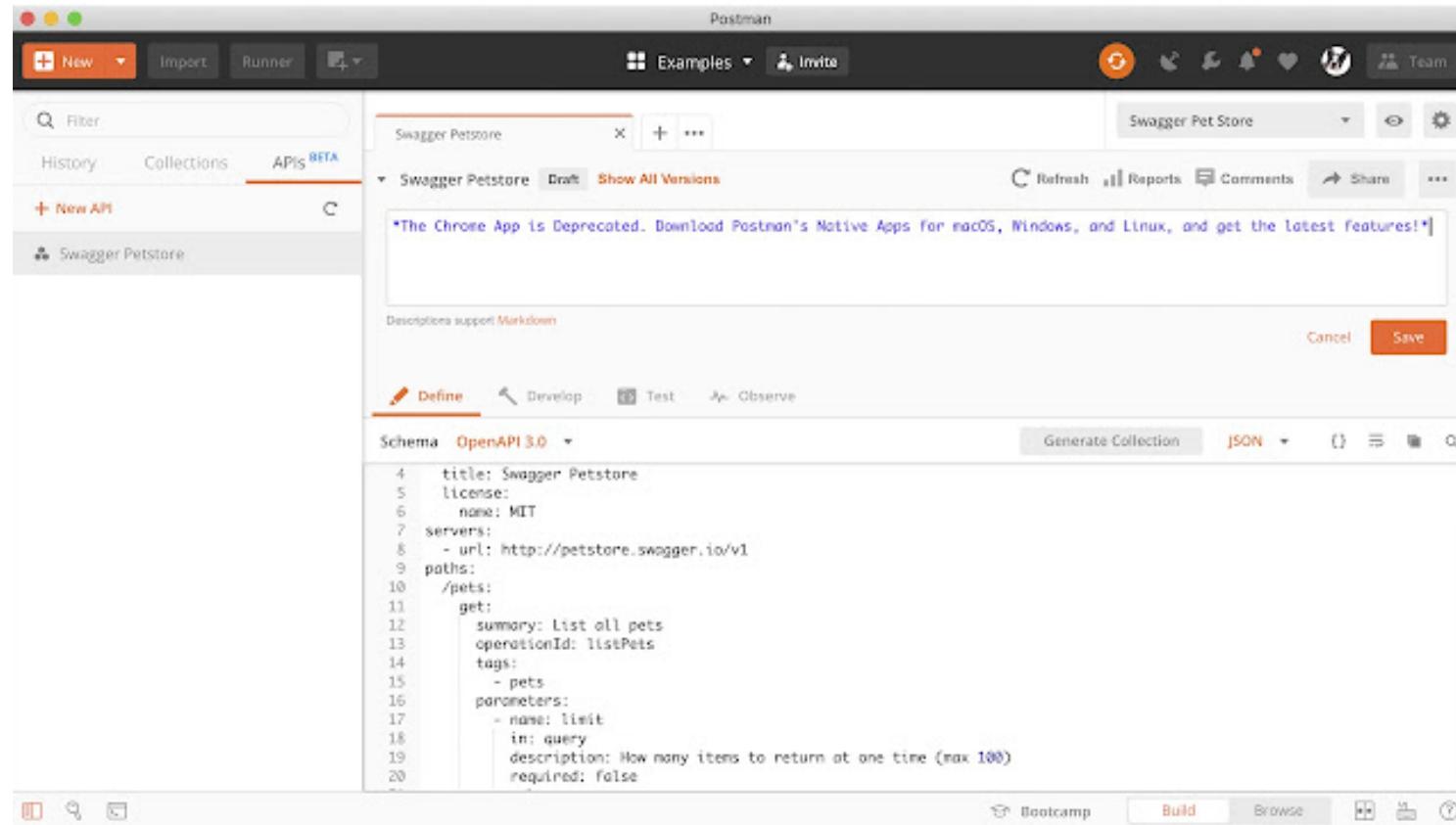
Q201. What is **Content Negotiation** in Web API? **V Imp**

Q202. What is **MediaTypeFormatter** class in Web API?

Q203. What are **Response Codes** in Web API?



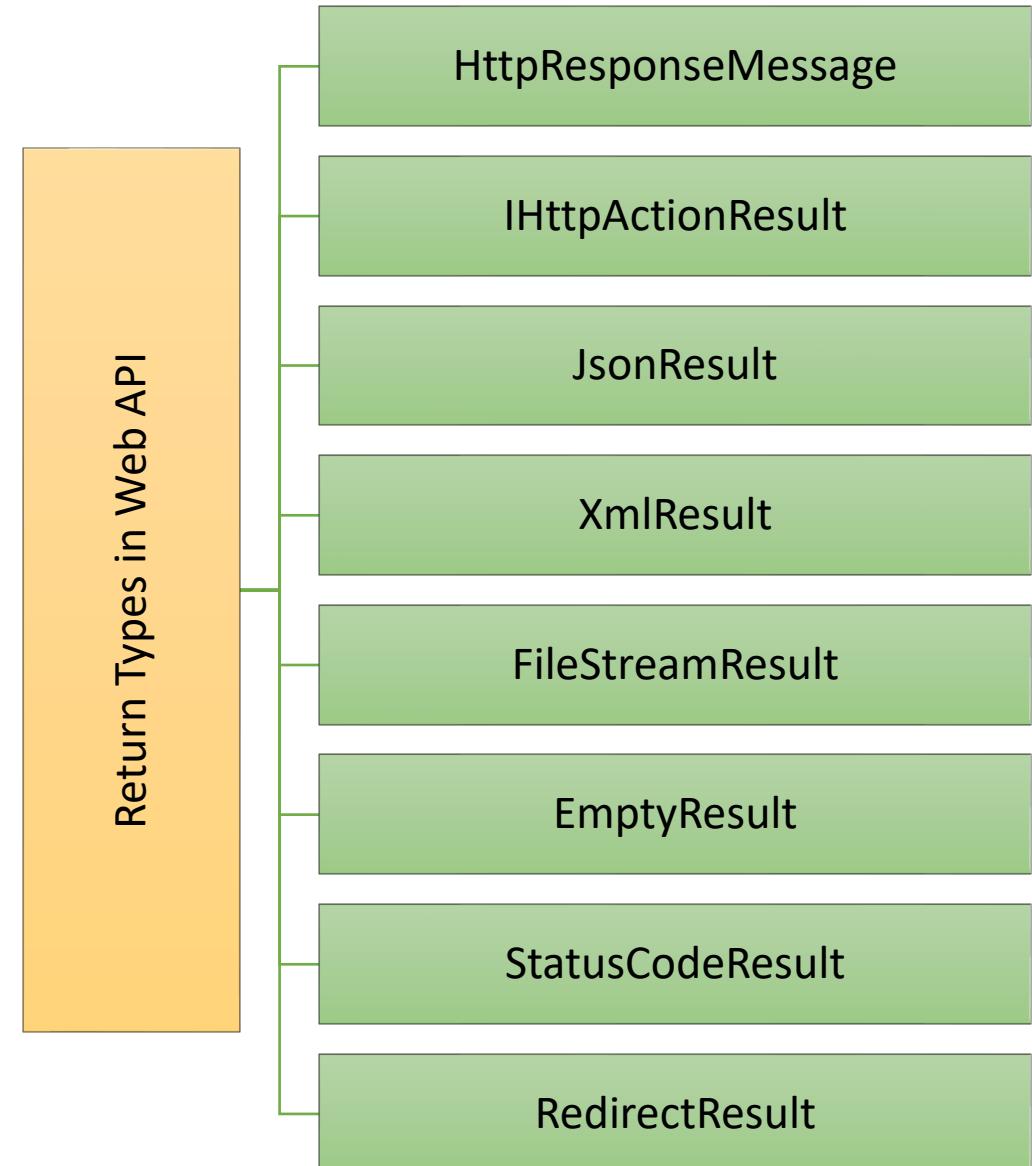
❖ Postman, Swagger and Fiddler tools



1. HttpResponseMessage: This class allows developers to create and return HTTP responses with a customized status code, headers, and content.

```
public HttpResponseMessage Get(int id)
{
    // create an HTTP response message with a status code
    // of 200 OK and the resource as the content
    var response = new HttpResponseMessage(HttpStatusCode.OK);
    response.Content = new StringContent("Resource found.");

    return response;
}
```

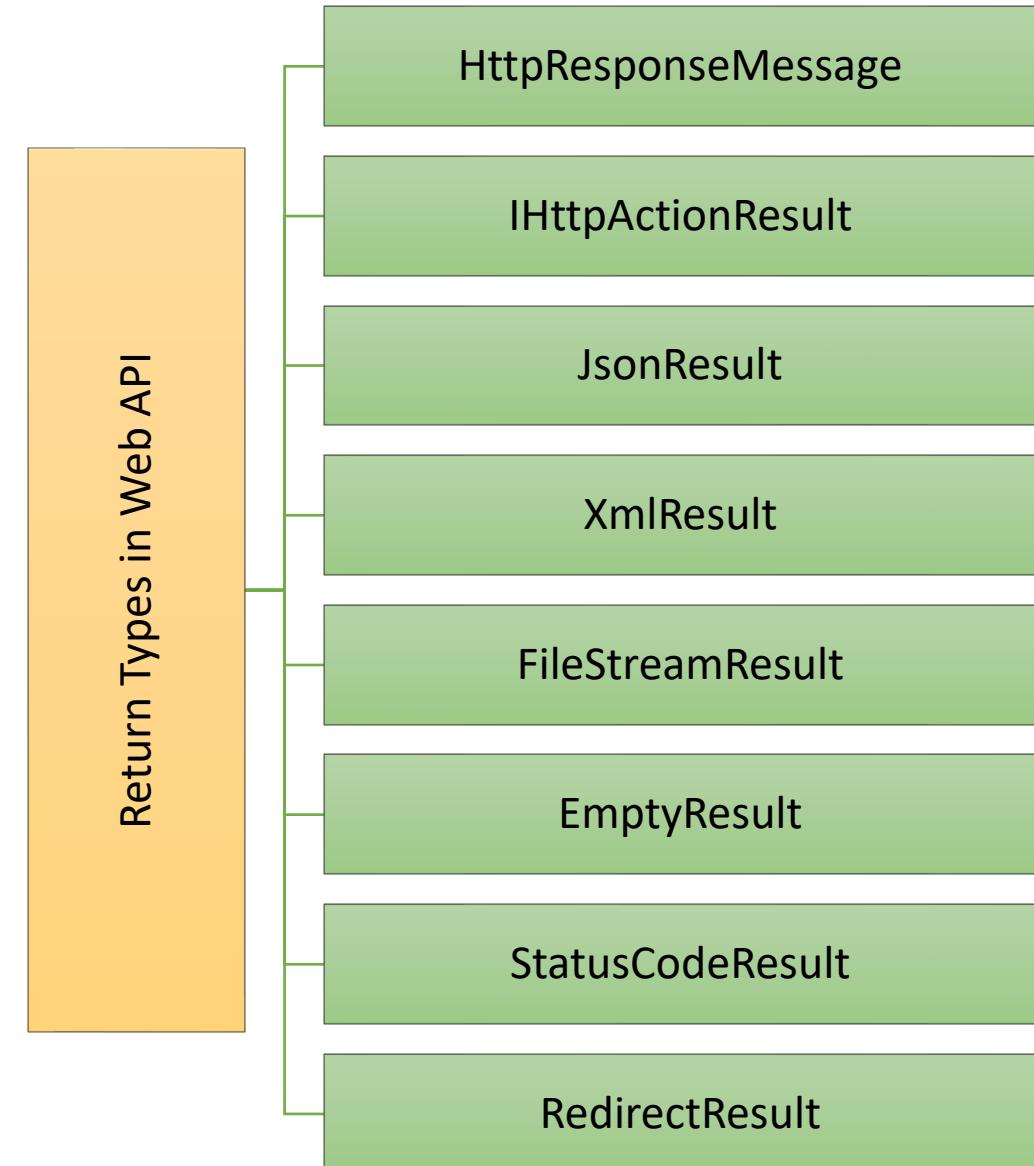


2. `IHttpActionResult`: This interface provides a level of abstraction between the controller and the HTTP response, allowing developers to return a variety of response types that implement the interface.

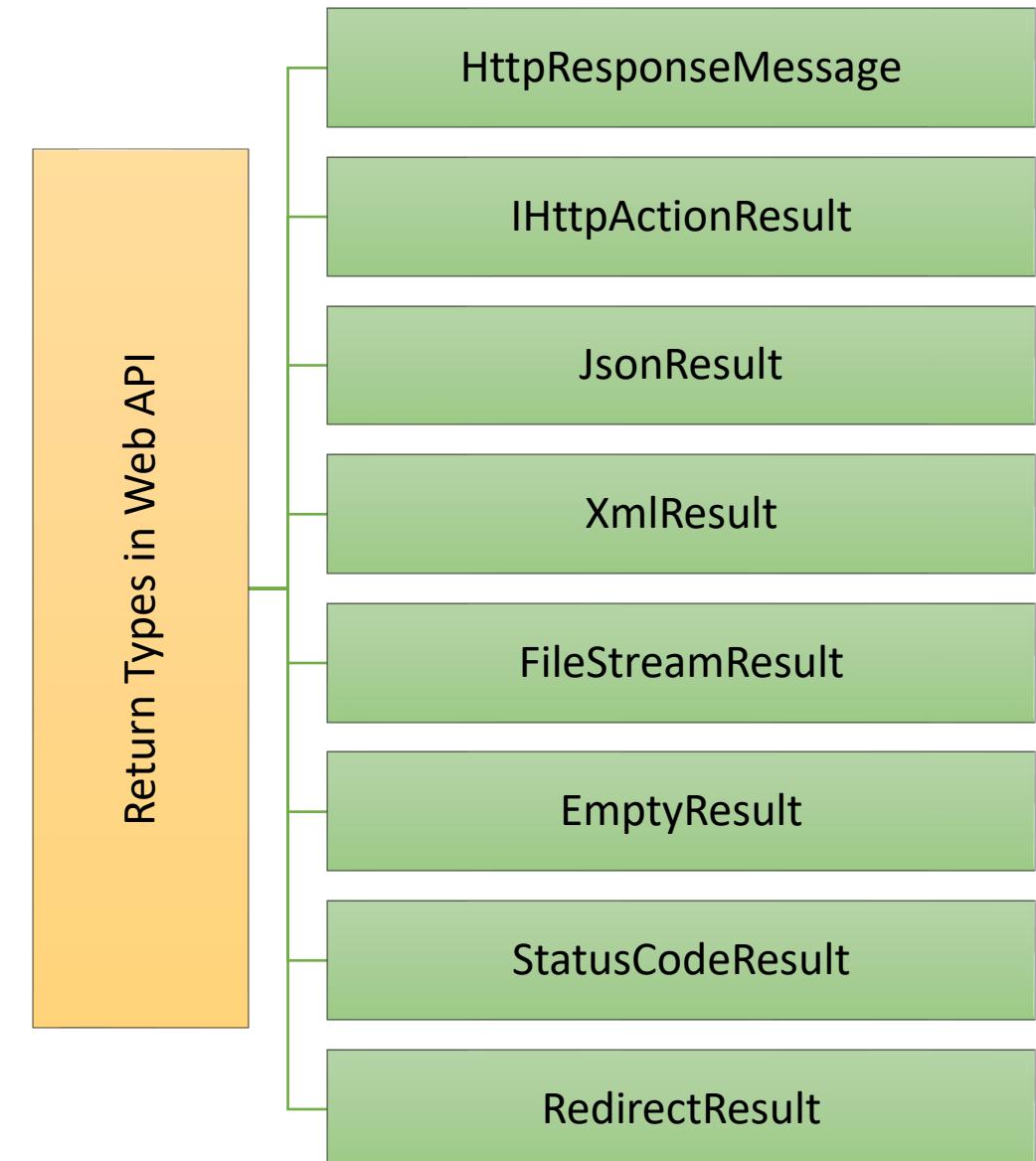
```
public IHttpActionResult Get(int id)
{
    // return an HTTP response with a status code
    // of 200 OK and the resource as the content
    return Ok("Resource found.");
}
```

3. `JsonResult`: This class is used to serialize an object to JSON format and return it as an HTTP response message.

```
public JsonResult Get(int id)
{
    // return the resource as JSON
    return Json(new { message = "Resource found." });
}
```



4. **XmlResult**: This class is used to serialize an object to XML format and return it as an HTTP response message.
5. **FileStreamResult**: This class is used to return a file as an HTTP response message by reading the file as a stream.
6. **EmptyResult**: This class is used to return an empty HTTP response message with a specified status code.
7. **StatusCodeResult**: This class is used to return an HTTP response message with a specified status code and no content.
8. **RedirectResult**: This class is used to redirect the client to a different URL by returning an HTTP response message with a redirect status code.



- ❖ In web API version 1.0, We have a response type class called **HttpResponseMessage** for returning Http response message from API.
- ❖ In Web API version 2.0 **IHttpActionResult/ IActionResult** is introduced which is basically the replacement of HttpResponseMessage.

```
[HttpGet]
public HttpResponseMessage Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return new HttpResponseMessage(HttpStatusCode.NotFound)
        {
            Content = new StringContent("Entity not found")
        };
    }
    else
    {
        var response = new HttpResponseMessage(HttpStatusCode.OK);
        response.Content = new ObjectContent<Entity>(entity, new
            JsonMediaTypeFormatter());
        return response;
    }
}
```



```
[HttpGet]
public IHttpActionResult Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return NotFound();
    }
    else
    {
        return Ok(entity);
    }
}
```

Benefits of IHttpActionResultResult:

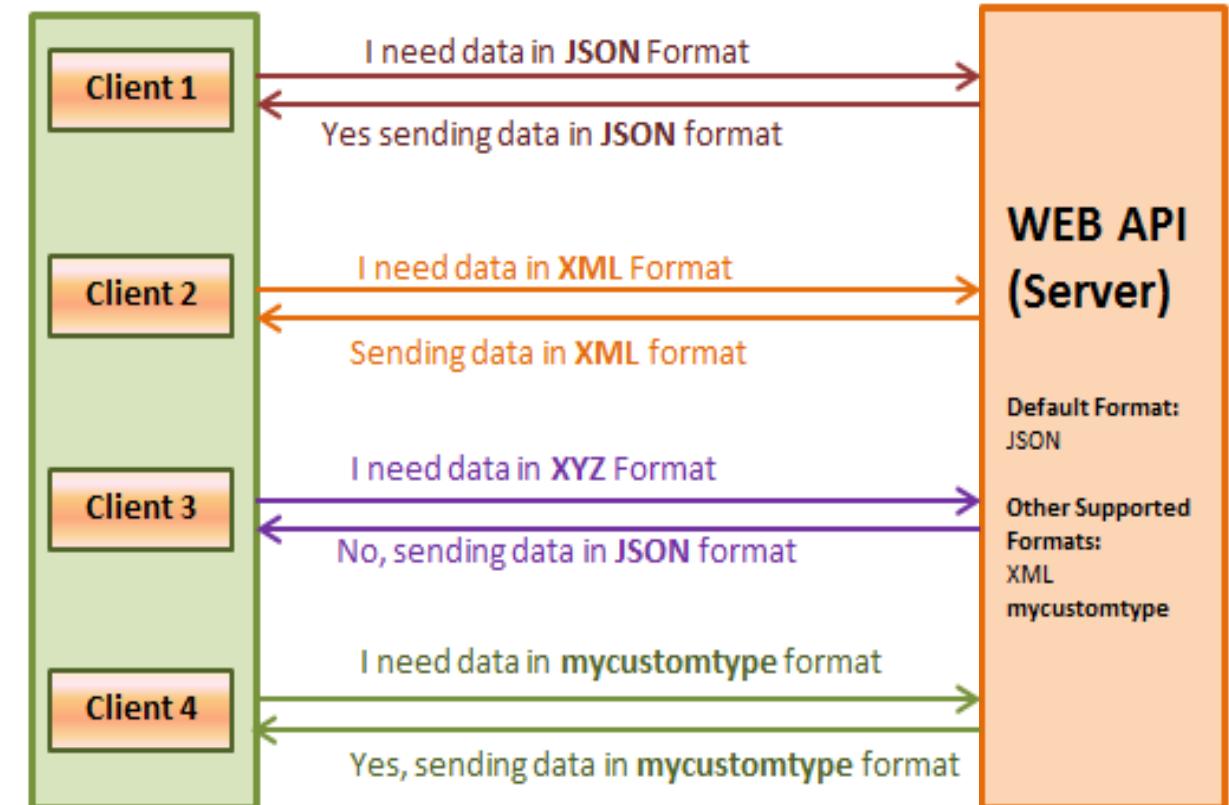
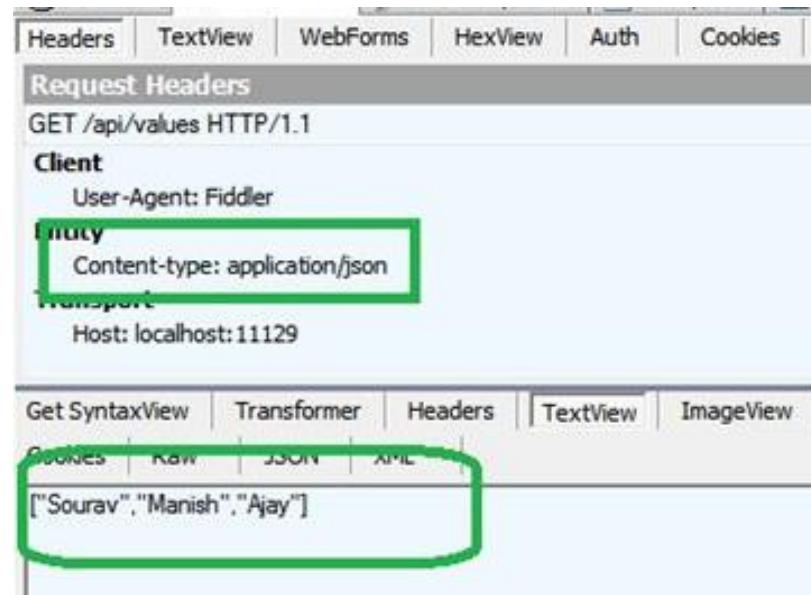
- 1. Readability:** The IHttpActionResultResult implementation is more readable as it uses simple NotFound() and Ok() methods.
- 2. Testability:** The IHttpActionResultResult implementation is more testable, as it allows for easier mocking of the return values.
- 3. Flexibility:** The IHttpActionResultResult implementation is more flexible, as it allows for easy customization of the HTTP response without needing to construct a HttpResponseMessage object manually.

- ❖ In Web API version 2.0 **IHttpActionResult/ActionResult** is introduced which is basically the replacement of HttpResponseMessage.

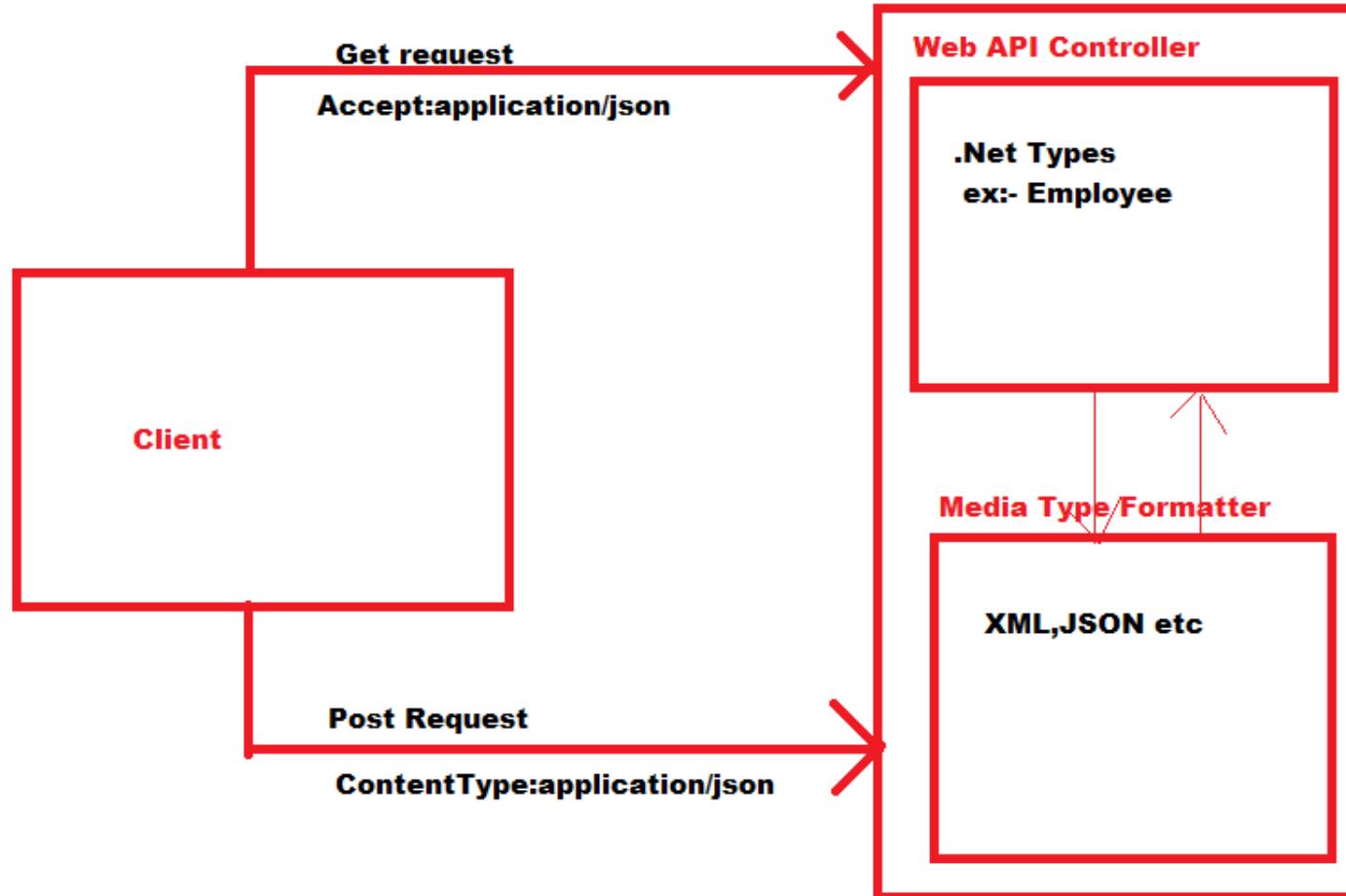
```
[HttpGet]
public IHttpActionResult Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return NotFound();
    }
    else
    {
        return Ok(entity);
    }
}
```

- ❖ `IHttpActionResult` is used in ASP.NET Web API, while `IActionResult` is used in ASP.NET Core.

- Content negotiation is a way for the client and server to agree on the format(json/ xml) of the data being exchanged.



- Content negotiation can be implemented with the help of MediaTypeFormatter class.



- ❖ MediaTypeFormatter is a class in ASP.NET Web API that handles the conversion of objects into serialized formats(JSON, XML, and other media) that can be transmitted over the web.

```
...public class JsonMediaTypeFormatter : BaseJsonMediaTypeFormatter
```

```
public abstract class BaseJsonMediaTypeFormatter : MediaTypeFormatter
```

```
// set the request headers to accept XML format
client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add
(new MediaTypeWithQualityHeaderValue("application/xml"));
```

1. **1xx: Informational** – Communicates transfer protocol-level information.
2. **2xx: Success** – Indicates that the client's request was accepted successfully.
3. **3xx: Redirection** – This means request is not complete. The client must take some additional action in order to complete their request.
4. **4xx: Client Error** – This means there is some error in API code.
5. **5xx: Server Error** – This means the error is not due to web api code but due to some environment settings.



- ❖ 200 OK: The request was successful, and the server has returned the requested data in the response body.
- ❖ 204 No Content: The request was successful, but the server has no data to return in the response body.
- ❖ 400 Bad Request: The client's request was invalid or could not be understood by the server.
- ❖ 401 Unauthorized: The client is not authorized to access the requested resource. The client should include authentication credentials in the request header to authenticate itself.
- ❖ 403 Forbidden: The client is authenticated, but not authorized to access the requested resource.
- ❖ 404 Not Found: The requested resource could not be found on the server.
- ❖ 500 Internal Server Error: An error occurred on the server while processing the request.



Chapter 24 : .NET Core - Basics

Q204. What is .NET Core?

Q205. What is .NET Standard?

Q206. What are the advantages of .NET Core over .NET framework? V Imp

Q207. What is the role of `Program.cs` file in ASP.NET Core?

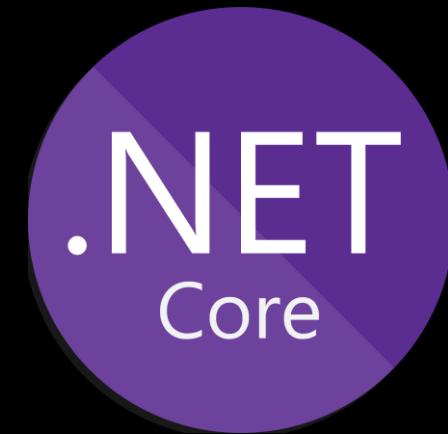
Q208. What is the role of `ConfigureServices` method?

Q209. What is the role of `Configure` method?

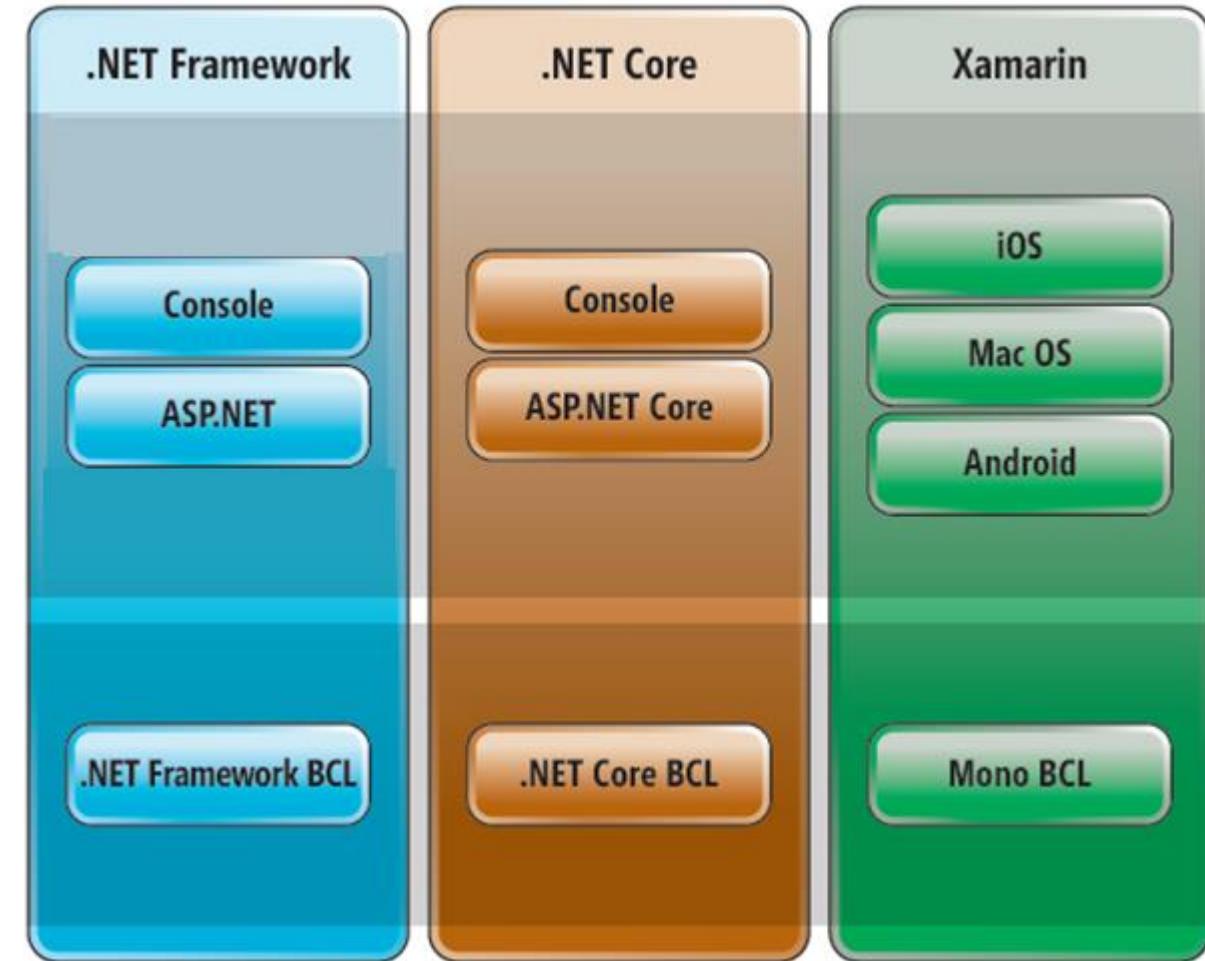
Q210. Describe the complete Request Processing Pipeline for ASP.NET Core MVC?

Q211. What is the difference between .NET Core and .NET 5?

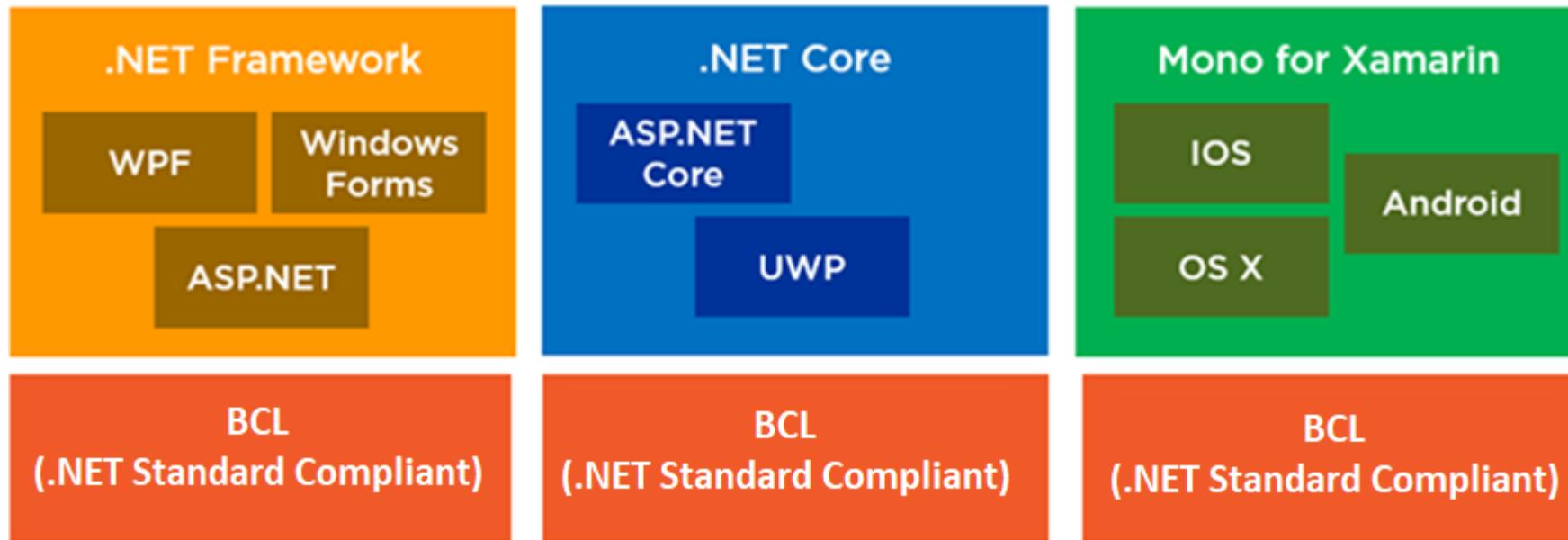
Q212. What is Metapackage? What is the name of Metapackage provided by ASP.NET Core?



- ❖ .NET Core is completely a **NEW** framework.
- ❖ It is **FREE** and **OPEN-SOURCE** platform.
- ❖ It is developed by Microsoft.



- ❖ .NET Standard is a set of rules that is common across all .NET frameworks.



CROSS PLATFORM

- Windows
- Linux
- MacOS

.NET Framework only supports Windows

OPEN SOURCE

- Free to use
- Modify
- Distribute

.NET Framework is paid

HOSTING

- Kestrel
- IIS
- Nginx

.NET Framework only support IIS Hosting

BUILT-IN DEPENDENCY INJECTION

- Loosely Coupled Design
- Reusability
- Testability

.NET framework don't have built in dependency injection

SUPPORT MULTIPLE IDE

- Visual Studio
- Visual Studio for Mac
- Visual Studio Code

.NET framework only support Visual Studio IDE

- ❖ Program.cs file contains the application startup code.
- ❖ The main method of Program.cs file is the entry point of application.

```
0 references
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    1 reference
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
    }
}
```

```
2 references
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime.
    // Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }

    // This method gets called by the runtime.
    // Use this method to configure the HTTP request pipeline.
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days.
            // You may want to change this for production scenarios, see https://aka.ms/aspnet/https
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
    }
}
```

- ❖ From .NET 6.0, the startup class code is merged in Program.cs file and therefore there is no separate Startup.cs class.

```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseAuthorization();
        app.MapControllers();
        app.Run();
    }
}
```

- ConfigureServices is used to add services to the application.
- ConfigureServices method always execute before Configure method.
- ConfigureServices is an optional method. It's not necessary that all its methods will be execute.



```

2 references
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime.
    // Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }
}

```

```

public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

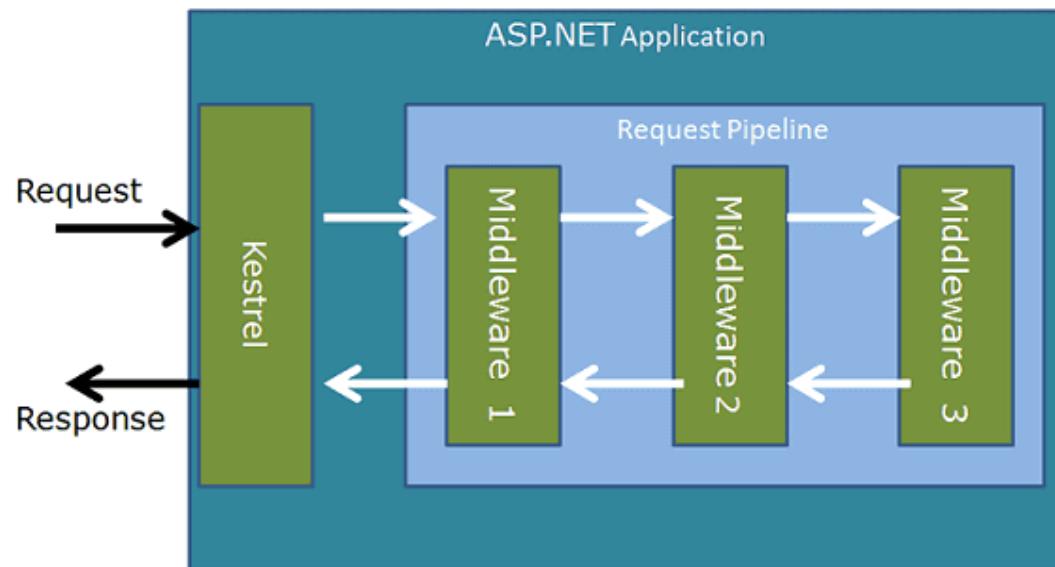
        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();
    }
}

```

[back to chapter index](#)

1. Configure method will configure the request pipeline.
2. Configure method executes after ConfigureServices method.
3. For each and every request, all the methods will execute inside the Configure method and in the same sequence.

Middleware in ASP.NET Request Pipeline



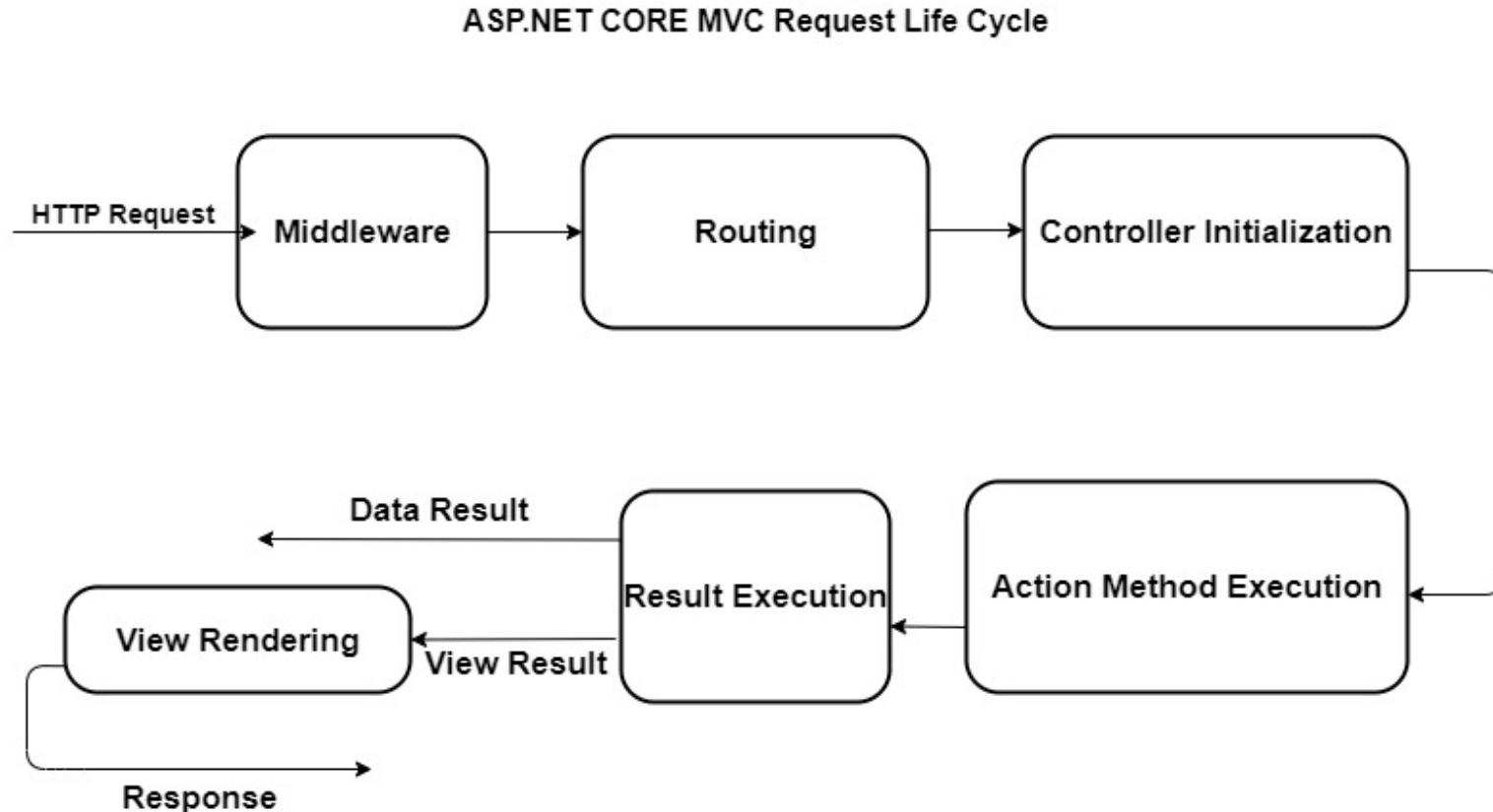
```
// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days.
        // You may want to change this for production scenarios, see https://go.microsoft.com/fwlink/?linkid=851014
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();
}
```

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();
```



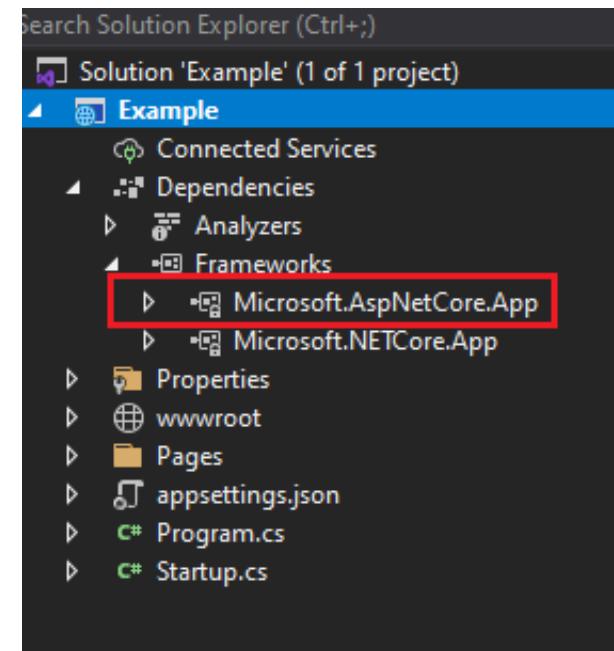
<http://localhost:1234/home/about>

[back to chapter index](#)

1. .NET 5 is the next major release after .NET Core 3.1.
2. The word 'Core' is dropped from the name to emphasize that .NET 5 is the future of all earlier versions of .NET Core.
3. Recently, Microsoft has introduced .NET 6.

- ❖ Metapackage is a consolidated package of many dependencies.

```
dependencies
  Microsoft.AspNetCore.Diagnostics
  Microsoft.AspNetCore.Hosting
  Microsoft.AspNetCore.Routing
  Microsoft.AspNetCore.Server.IISIntegration
  Microsoft.AspNetCore.Server.Kestrel
  Microsoft.Extensions.Configuration.EnvironmentVariables
  Microsoft.Extensions.Configuration.FileExtensions
  Microsoft.Extensions.Configuration.Json
  Microsoft.Extensions.Logging
  Microsoft.Extensions.Logging.Console
  Microsoft.Extensions.Options.ConfigurationExtensions
  NETStandard.Library
```



Chapter 25 : .NET Core - Dependency Injection

Q213. What is **Dependency Injection**? **V Imp**

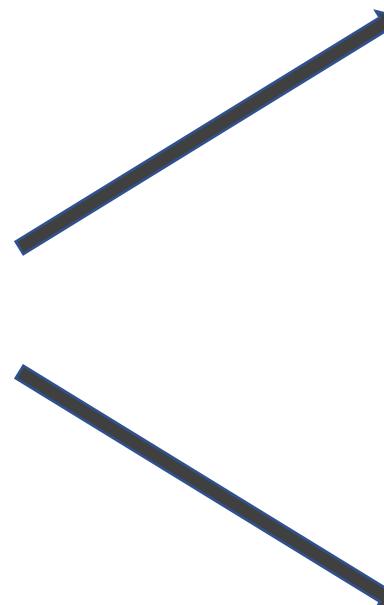
Q214. How to **implement** Dependency injection in .NET Core? **V Imp**

Q215. What are the **advantages** of Dependency Injection in .NET Core? **V Imp**

Q216. How to **use** Dependency Injection in Views in ASP.NET Core?

- ❖ Dependency Injection (DI) is a software **design pattern** in which we inject the dependency object of a class into another class.

```
public class Salary
{
    public int CalculateSalary()
    {
        return 1000000;
    }
}
```



```
public class Employee
{
    public void GetSalary()
    {
        Salary sal = new Salary();

        int salary = sal.CalculateSalary();
    }
}
```

```
public class Employee
{
    private readonly Salary _salary;

    public Employee(Salary salary)
    {
        _salary = salary;
    }

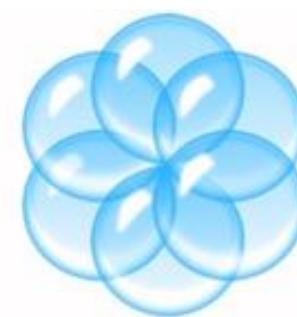
    public void GetSalary()
    {
        int salary = _salary.CalculateSalary();
    }
}
```

```
0 references
public class FirstController : Controller
{
    0 references
    public int Index()
    {
        //Create object of math class
        MathStudent cls = new MathStudent();
        return cls.GetStudentCount();
    }
}
```



```
4 references
public class MathStudent
{
    2 references
    public int GetStudentCount()
    {
        return 50;
    }
}
```

```
0 references
public class SecondController : Controller
{
    0 references
    public int Index()
    {
        //Create object of math class
        MathStudent cls = new MathStudent();
        return cls.GetStudentCount();
    }
}
```



Tight Coupling

```
1 reference
public class ScienceStudent
{
    0 references
    public int GetStudentCount()
    {
        return 100;
    }
}
```

100 Controllers
(Big/Enterprise Level Application)

[back to chapter index](#)

```
1 reference
public class FirstController : Controller
{
    private IStudent _student;
    0 references
    public FirstController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```

```
1 reference
public class SecondController : Controller
{
    private IStudent _student;
    0 references
    public SecondController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```



```
1 reference
public interface IStudent
{
    0 references
    public int GetStudentCount();
}
```

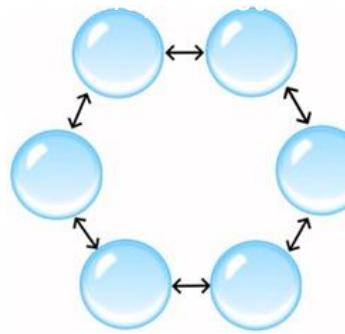
```
4 references
public class MathStudent : IStudent
{
    3 references
    public int GetStudentCount()
    {
        return 50;
    }
}
```

```
1 reference
public class ScienceStudent : IStudent
{
    1 reference
    public int GetStudentCount()
    {
        return 100;
    }
}
```

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IStudent, ScienceStudent>();
}
```



Tight Coupling



Loose Coupling

- 1. Flexibility:** DI allows you to easily change the behavior of an application without modifying its code. By injecting dependencies, you can easily switch between different implementations of a service/ class.
- 2. Easier unit testing:** DI makes it easy to unit test your code by allowing you to easily replace real dependencies with mock with the help of interfaces.
- 3. Independent modules:** By separating dependencies, it becomes easier to make changes to your code without affecting the rest of the system.
- 4. Reusability:** DI promotes reuse of components by making them independent of their environment. This allows you to reuse the same components in different applications without modification.

- ❖ A service can be injected into a view using the **@inject directive**.

```
public interface IStudent
{
    2 references
    public int GetStudentCount();
}
```

```
public class MathStudent:IStudent
{
    2 references
    public int GetStudentCount()
    {
        return 100;
    }
}
```

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();

    services.AddScoped<IStudent, MathStudent>();
}
```

```
@inject WebApplication1.IStudent _student

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    Total Student: @_student.GetStudentCount();
</div>
```

```
0 references
public class FirstController : Controller
{
    private IStudent _student;
    0 references
    public FirstController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```

Chapter 26 : .NET Core - Service Lifetimes, Middleware & Hosting

Q217. What are the types of [Service Lifetimes](#) of an object in ASP.NET Core?

Q218. What is [AddSingleton](#), [AddScoped](#) and [AddTransient](#) method? V Imp

Q219. What is [Middleware](#) in ASP.NET Core? What is [custom middleware](#)? V Imp

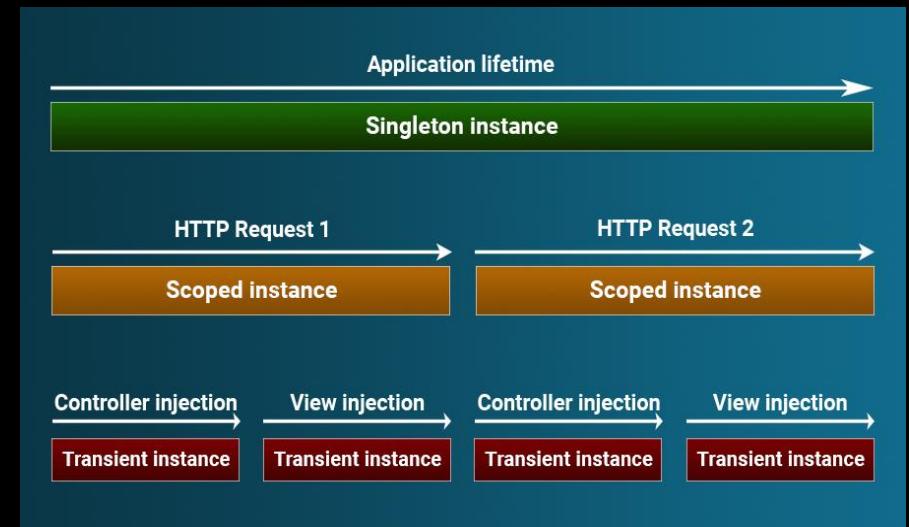
Q220. How [ASP.NET Core](#) [Middleware](#) is different from [HttpModule](#)?

Q221. What is [Request Delegate](#) in .NET Core?

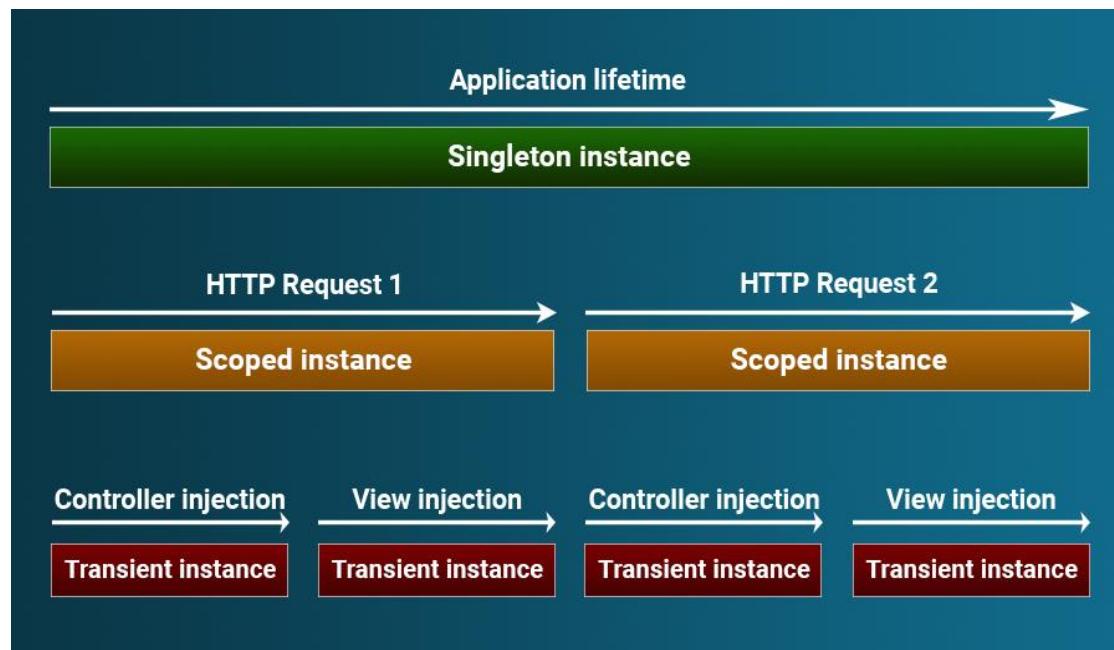
Q222. What is [Run\(\)](#), [Use\(\)](#) and [Map\(\)](#) method? V Imp

Q223. What are the [types of Hosting](#) in ASP.NET Core? What is In process and Out of process hosting?

Q224. What is [Kestrel](#)? What is the difference between Kestrel and IIS?



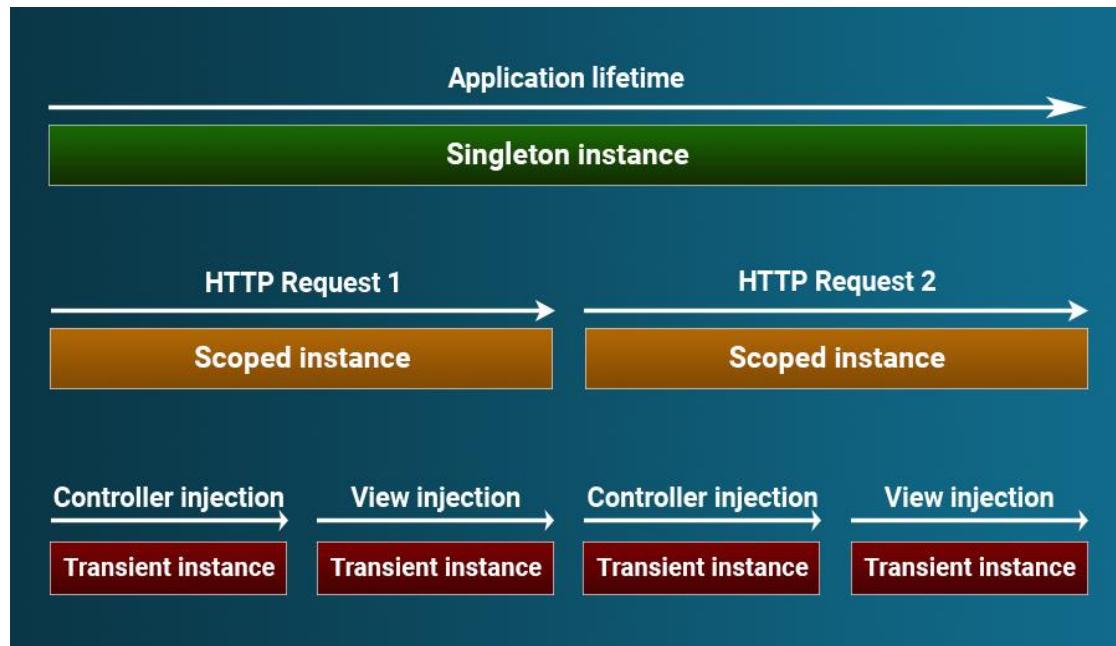
- ❖ The Service Lifetime refers to the lifespan of a registered service. Or
- ❖ The service lifetime controls how long a result object will live for after it has been created by the container.



```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IStudent, ScienceStudent>();
}
```

```
1 reference
public class FirstController : Controller
{
    private IStudent _student;
    public FirstController(IStudent student)
    {
        _student = student;
    }
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```

- ❖ AddSingleton method create only **one instance** when the service is requested for first time. And then the same instance will be shared by all different http requests.

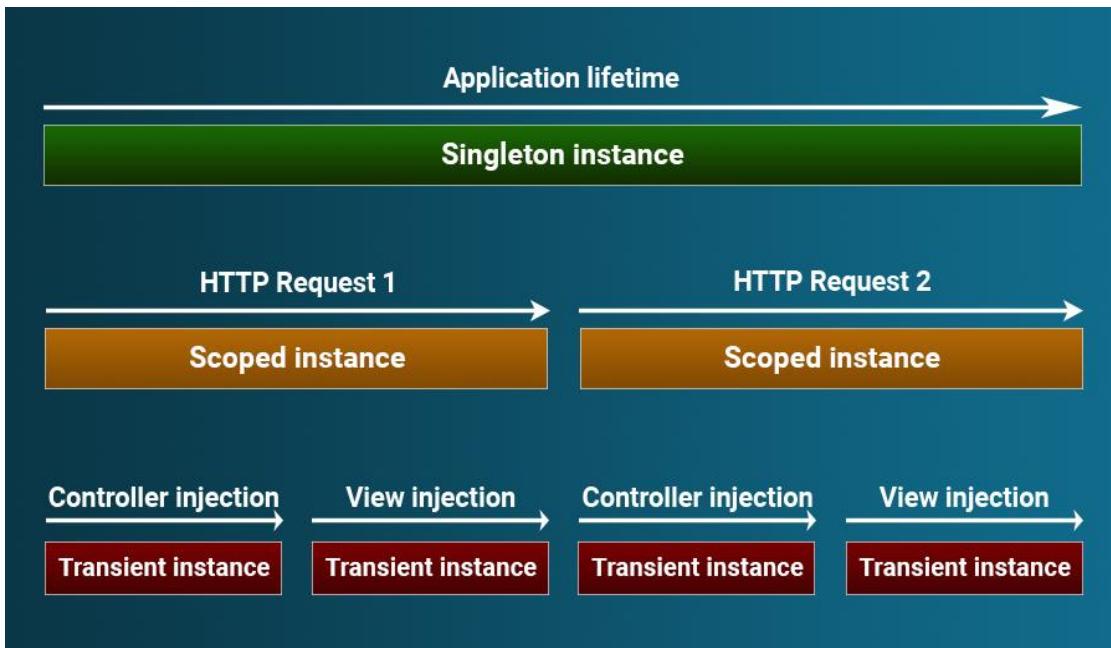


```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<ILogger, Logger>();
    //services.AddSingleton<IPayment, Payment>();
    //services.AddTransient<IDataAccess, DataAccess>();

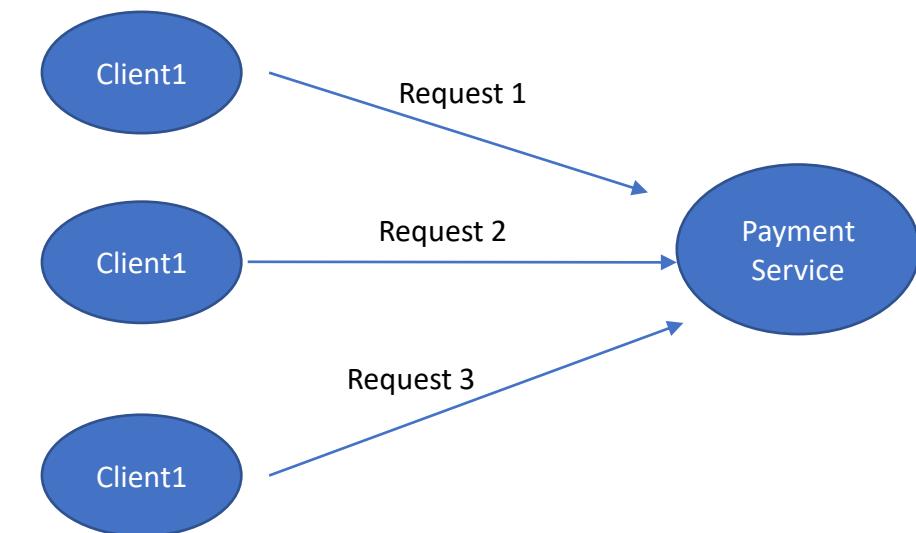
}
```

```
Jul 6 08:42:01 ip-10-17-12-120 rsyslogd: [origin software="rsyslog.com"] rsyslogd was HUPed
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: bound to
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPACK
Jul 6 23:42:42 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 7 08:44:47 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 7 19:41:23 ip-10-17-12-120 dhclient[2486]: bound to
Jul 8 02:47:00 ip-10-17-12-120 yum[31369]: Installed: p
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPACK
Jul 8 07:06:13 ip-10-17-12-120 dhclient[2486]: bound to
```

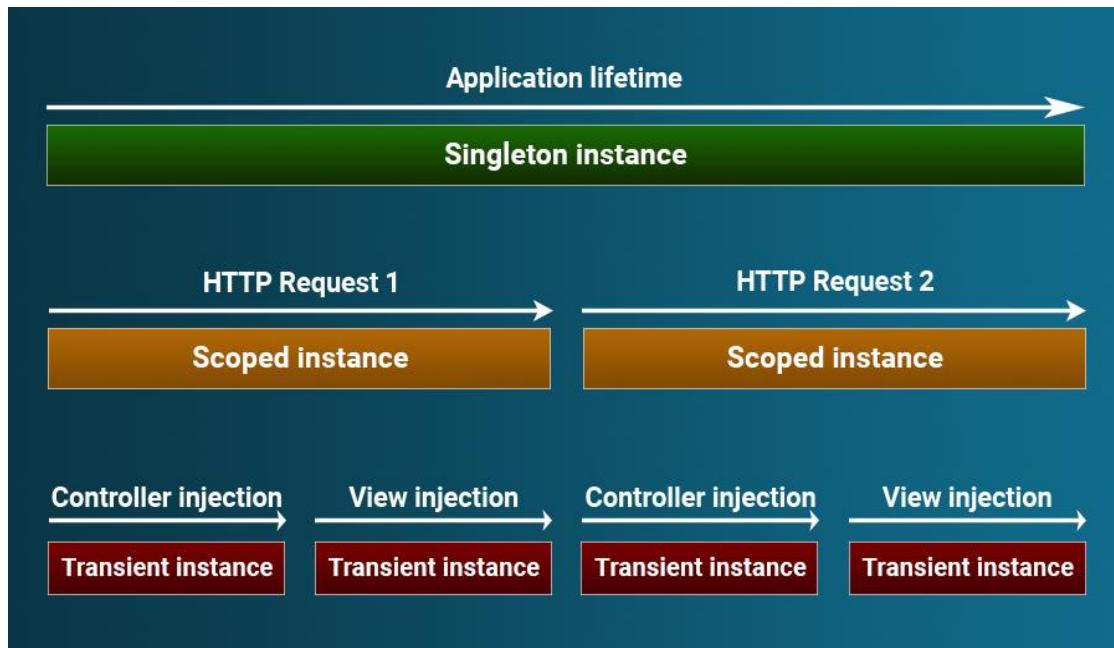
- ❖ AddScoped method create single instance per request. For every individual request there will be a single instance or object.



```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    //services.AddSingleton<ILogger, Logger>();
    services.AddScoped<IPayment, Payment>();
    //services.AddTransient<IDataAccess, DataAccess>();
}
```

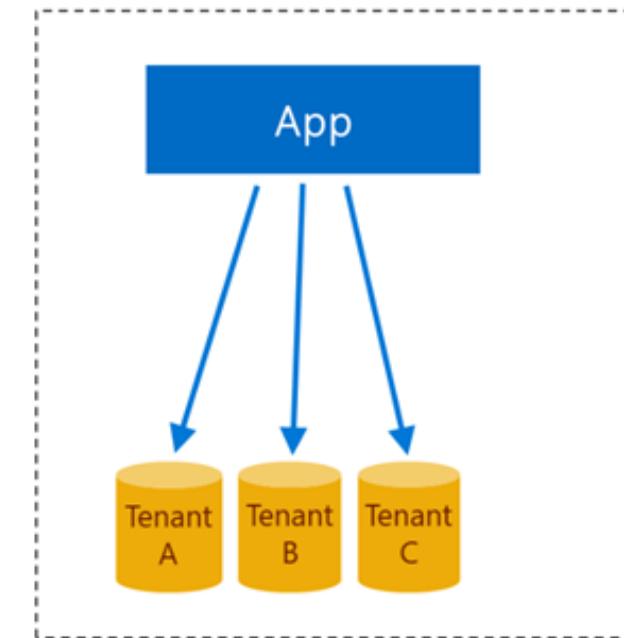


- ❖ AddTransient instance will not be shared at all, even with in the same request. Every time a new instance will be created.

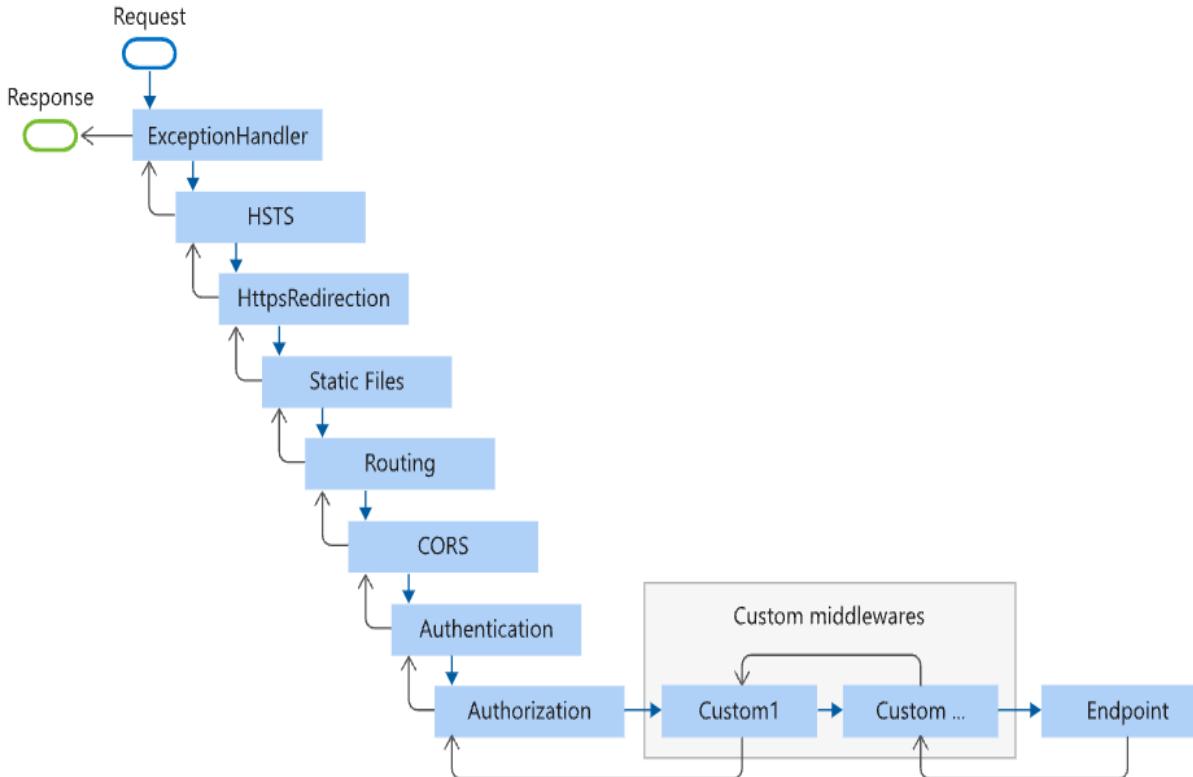


```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    //services.AddSingleton<ILogger, Logger>();
    //services.AddScoped<IPayment, Payment>();
    services.AddTransient<IDataAccess, DataAccess>();

}
```



- ❖ A middleware is a component that is executed on every request in the ASP.NET Core application.



```

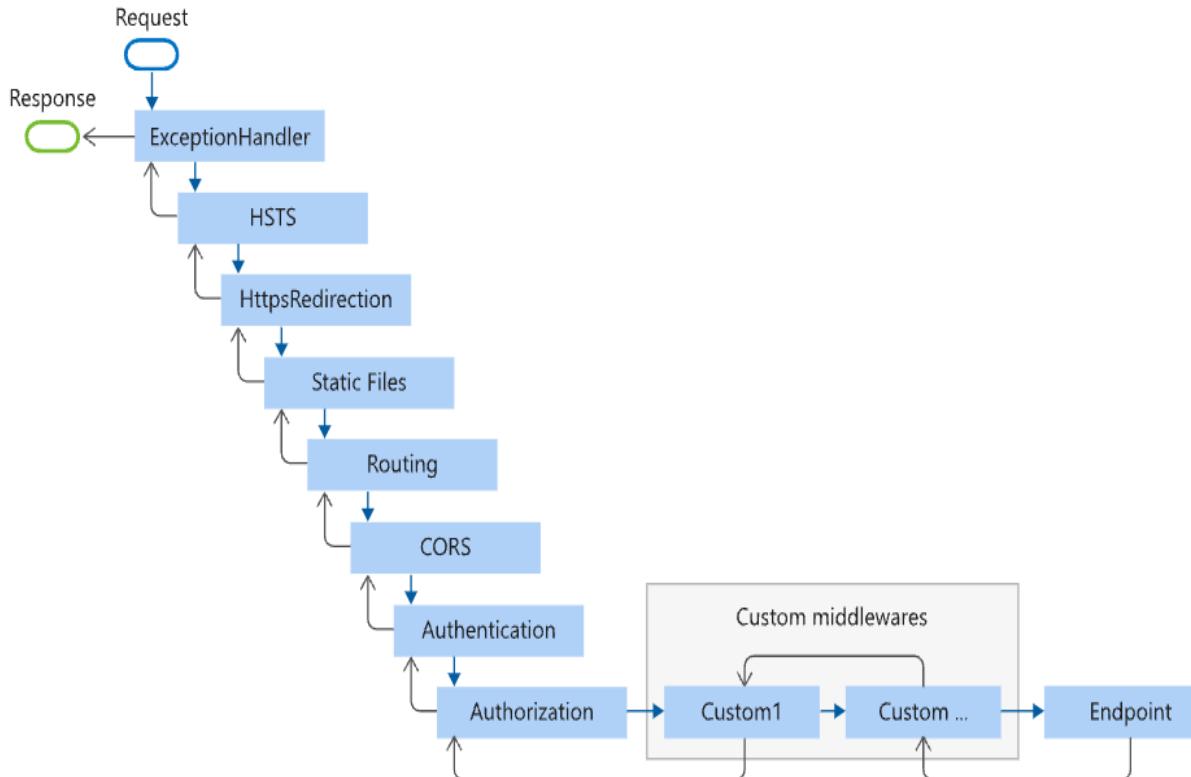
// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
  
```

- ❖ A middleware a component that is executed on **EVERY REQUEST** in the ASP.NET Core application.
- ❖ We can set up the middleware in ASP.NET using the **CONFIGURE** method of our **STARTUP** class.



```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

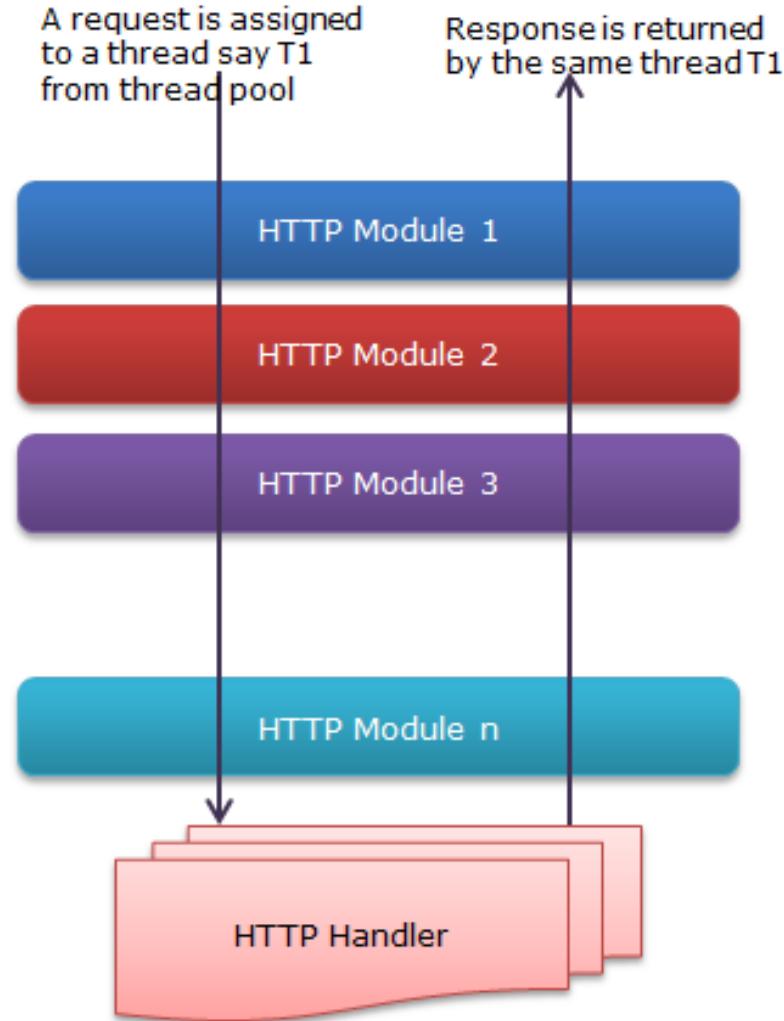
        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

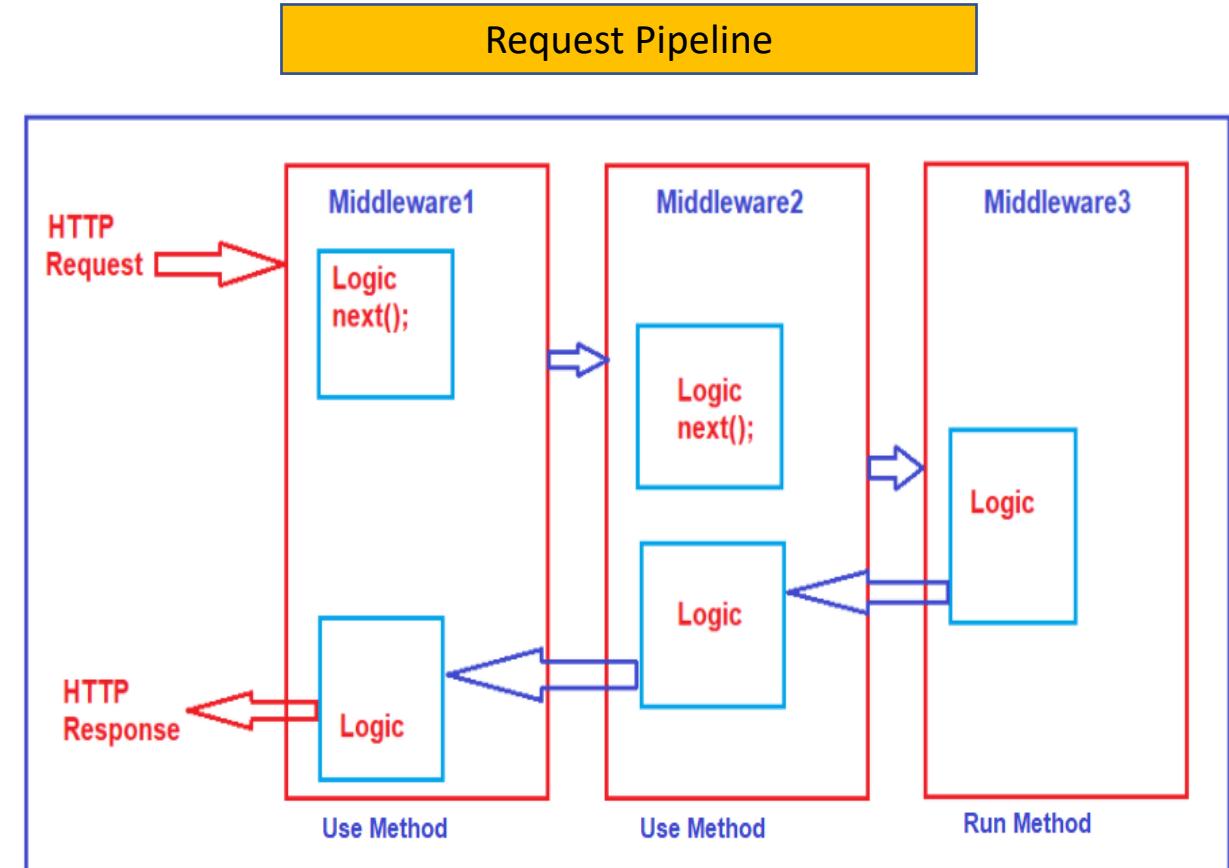
        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseAuthorization();
        app.MapControllers();
        app.Run();
    }
}
```

- ❖ HttpModules are nothing else, but they are very similar to Middlewares only.
- ❖ HttpModules are registered in the web.config or global.asax file of the ASP.NET framework, while a Middleware is registered via Configure method of the startup.cs class.

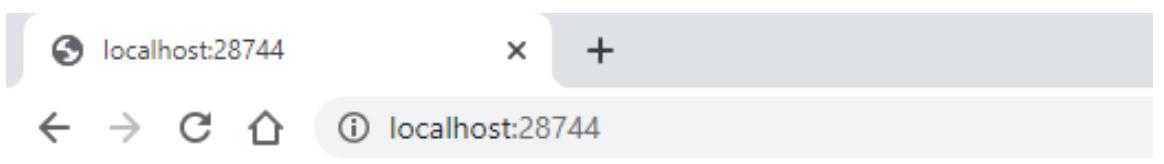


- ❖ Request delegates are used to **build** the request pipeline.
- ❖ Request delegates are configured using **Run**, **Map**, and **Use** extension methods.



- ❖ For any request, Use method will execute the middleware component and then pass the execution to the **next** middleware or component.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync("Hello from 1st delegate.");
        await next();
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 2nd Middleware");
    });
}
```



Hello from 1st delegate.Hello from 2nd Middleware

```
// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }
    app.UseStaticFiles();

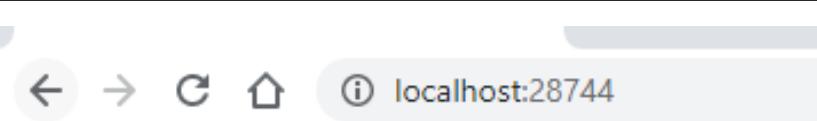
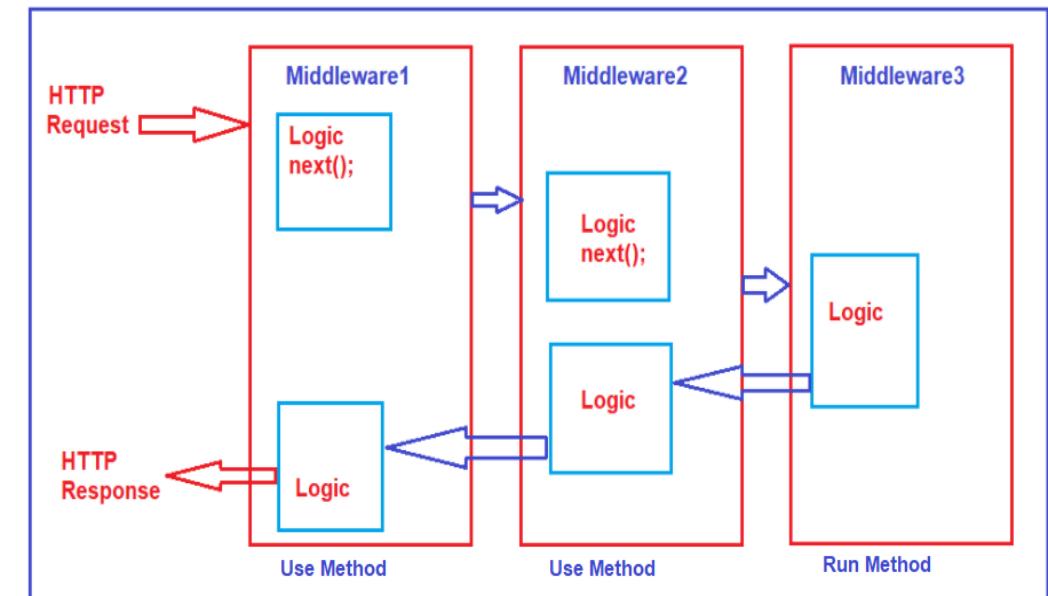
    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

- ❖ Run method will execute the middleware component and then **terminate** the execution.
- ❖ It should be placed at the end of any pipeline.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Hello from 1st delegate.");
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 2nd Middleware");
    });
}
```

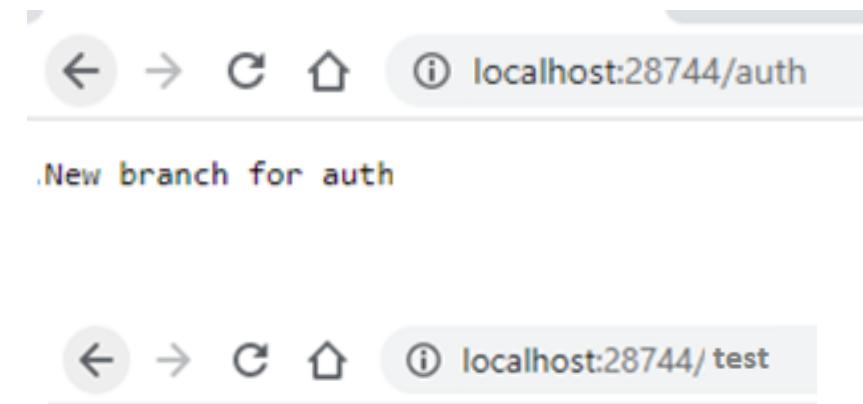


Hello from 1st delegate.

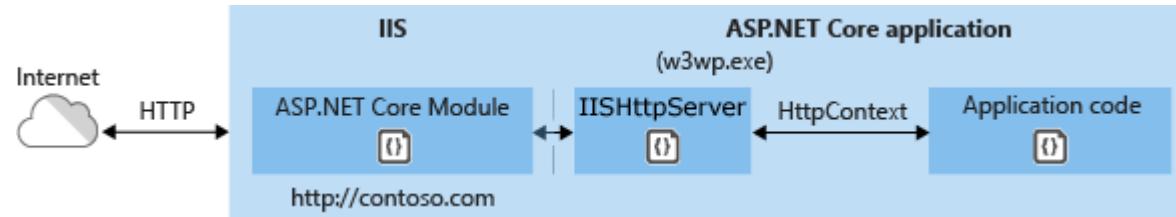
[back to chapter index](#)

- ❖ The Map method is used to **map** a specific request url path to a middleware component.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Map("/auth", a =>
    {
        a.Run(async (context) =>
        {
            await context.Response.WriteAsync("New branch for auth");
        });
    });
}
```

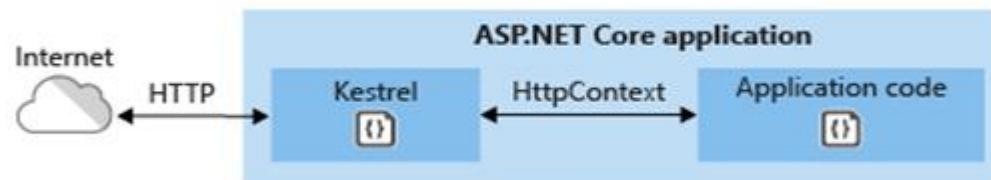


❖ In Process Hosting

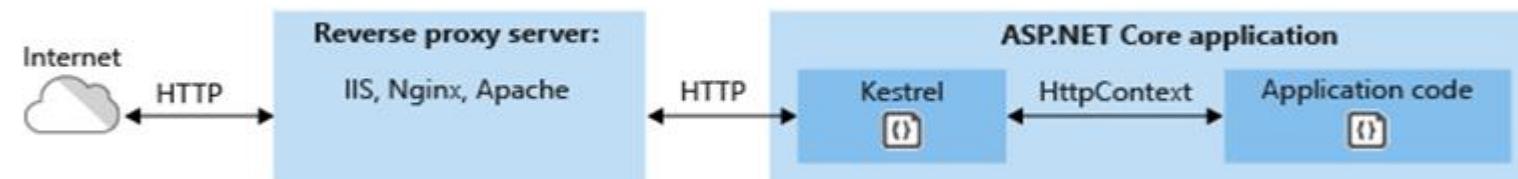


❖ Out of Process Hosting

Kestrel used as an edge (Internet-facing) web server:



Kestrel used in a reverse proxy configuration:



In-Process	Out of Process
Process name is <code>w3wp.exe</code>	Process name is <code>dotnet.exe</code>
Only one web server (IIS)	Two web server (IIS/Nginx/Apache + Kestrel) One web server (Kestrel)

- ❖ Kestrel is a **lightweight cross-platform** web server for asp.net core.
- ❖ It is a type of out of process hosting. It can be used alone, or it can be used with IIS or Nginx web servers.

Kestrel	IIS
1. Kestrel is a lightweight web server used for hosting.	IIS is a complete web server which is also used for hosting.
2. Kestrel is cross platform and can be used with other web servers like IIS, Nginx and Apache.	IIS is not cross platform and can only run in windows.
3. Kestrel is open source like .NET Core	IIS is not open source

Chapter 27 : .NET Core - Routing, Files, CORS & More...

Q225. What is **Routing**? Explain attribute routing in ASP.NET Core? V Imp

Q226. Explain **default project structure** in ASP.NET Core application?

Q227. How ASP.NET Core serve **static files**?

Q228. What are the roles of **Appsettings.Json** and **Launchsetting.Json** file?

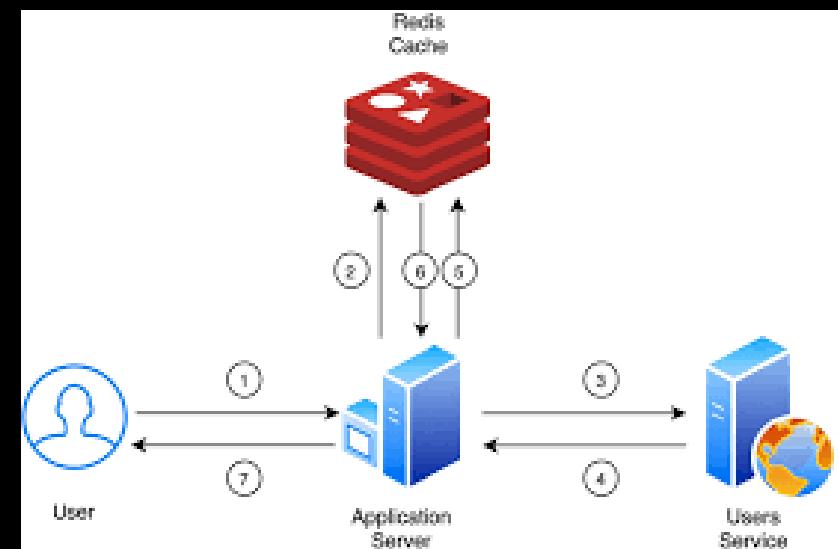
Q229. What are the various techniques to **save configuration** settings in .NET Core?

Q230. What is **CORS**? Why is CORS restriction is required? How to fix CORS error? V Imp

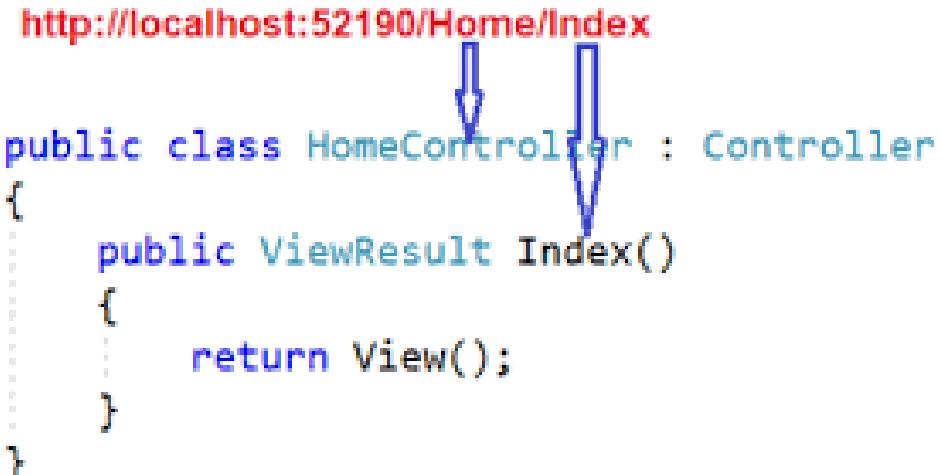
Q231. What is **In-Memory caching & Distributed Caching**?

Q232. How to handle **errors** in ASP.NET Core?

Q233. What are **Razor pages** in .NET Core?



- ❖ Routing in MVC is the process of mapping a URL request to a specific controller action in a web application.
- ❖ The routing system is responsible for directing incoming requests to the appropriate controller, based on the URL.
- ❖ The routing system is typically configured using a **routing table**, which maps URLs to controller actions.



http://localhost:52190/Home/Index

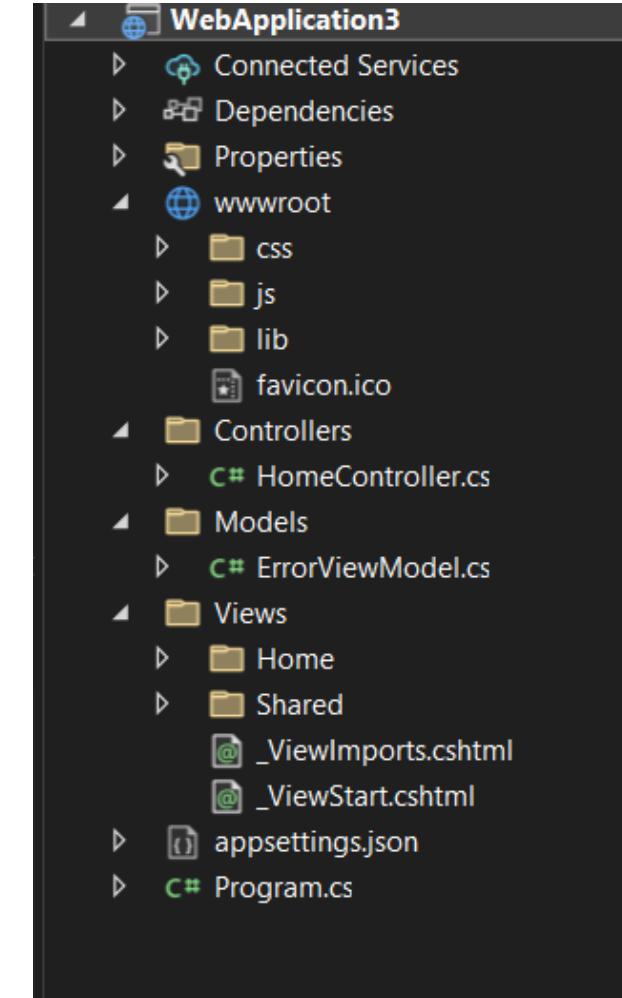
```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        return View();
    }
}
```

- ❖ Attribute based routing is used to manipulate the default behavior of routing in ASP.NET MVC.

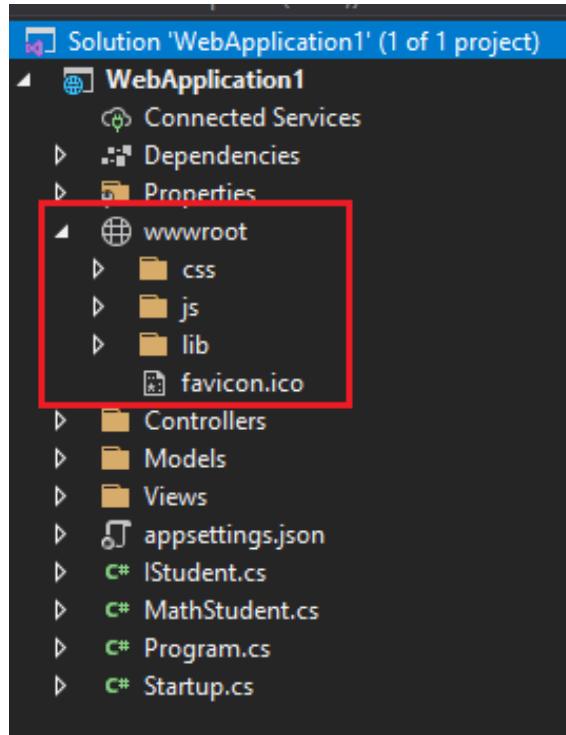
`http://localhost:1234/home/about`

```
public class HomeController: Controller
{
    [Route("Users/about")]
    Public ActionResult GotoAbout()
    {
        return View();
    }
}
```

- ❖ **wwwroot** - Store static files of the application like js/css/images.
- ❖ **Appsettings.json** – Configuration settings like database connection strings and other things are saved here.
- ❖ **Program.cs** – Entry point of the application.



- ❖ WWWROOT contains all the static files.
- ❖ USESTATICFILES() enables the static files to be served to client.



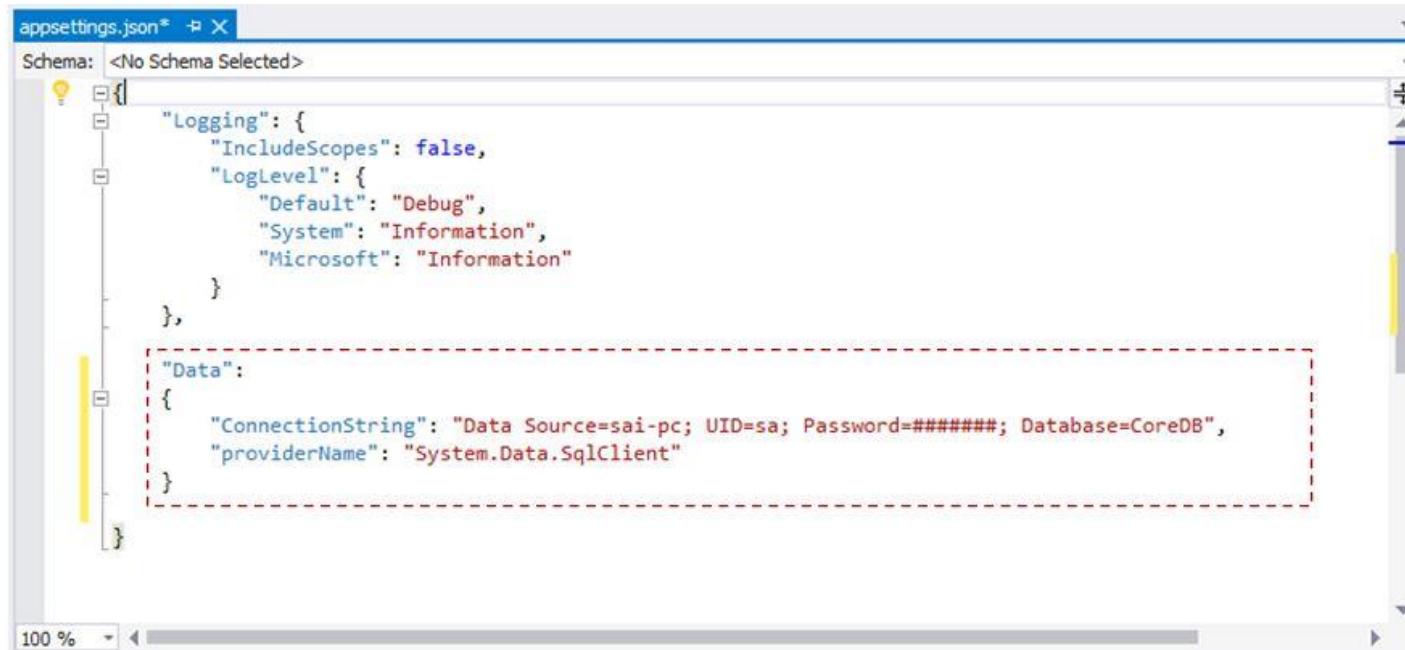
```
// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

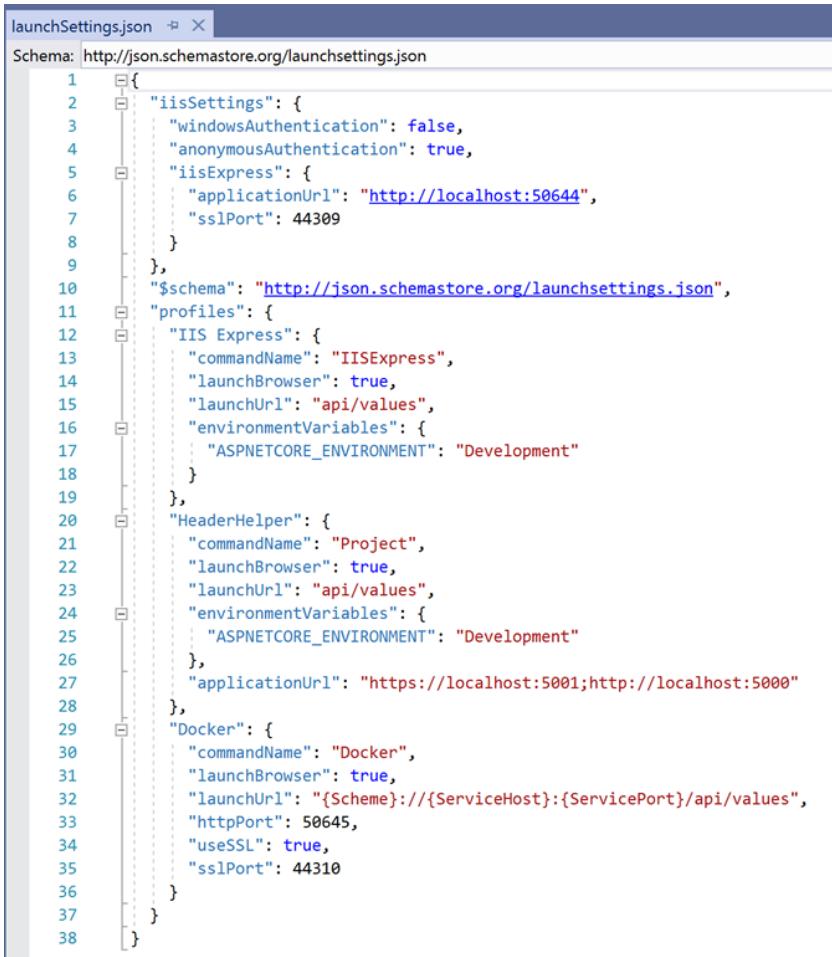
- ❖ The appsettings.json file is an application configuration file used to **store configuration settings** such as database connections strings etc.



```
appsettings.json*  # X
Schema: <No Schema Selected>
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "Data": {
    "ConnectionString": "Data Source=sai-pc; UID=sa; Password=#####; Database=CoreDB",
    "providerName": "System.Data.SqlClient"
  }
}
```



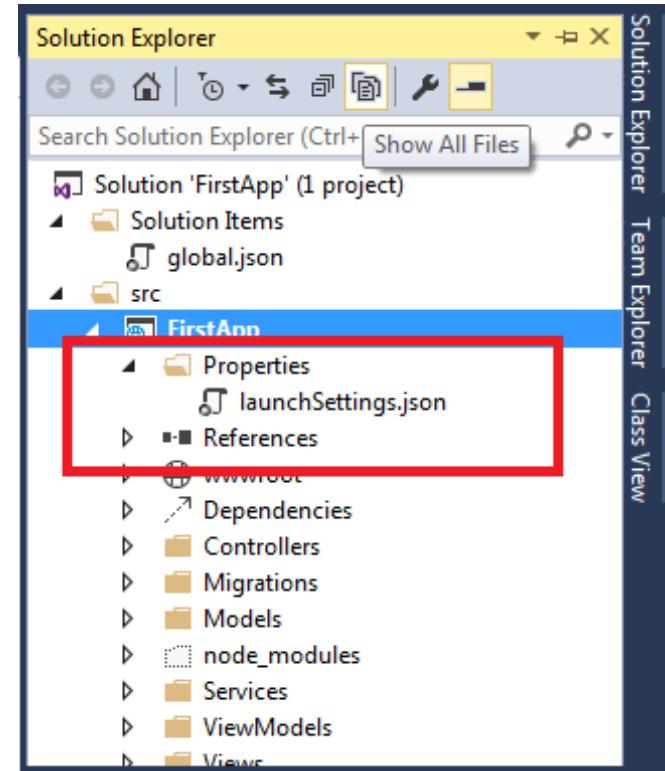
- ❖ The launchSettings. json file is used to store the configuration information, which is used to **start** the ASP.NET Core application.



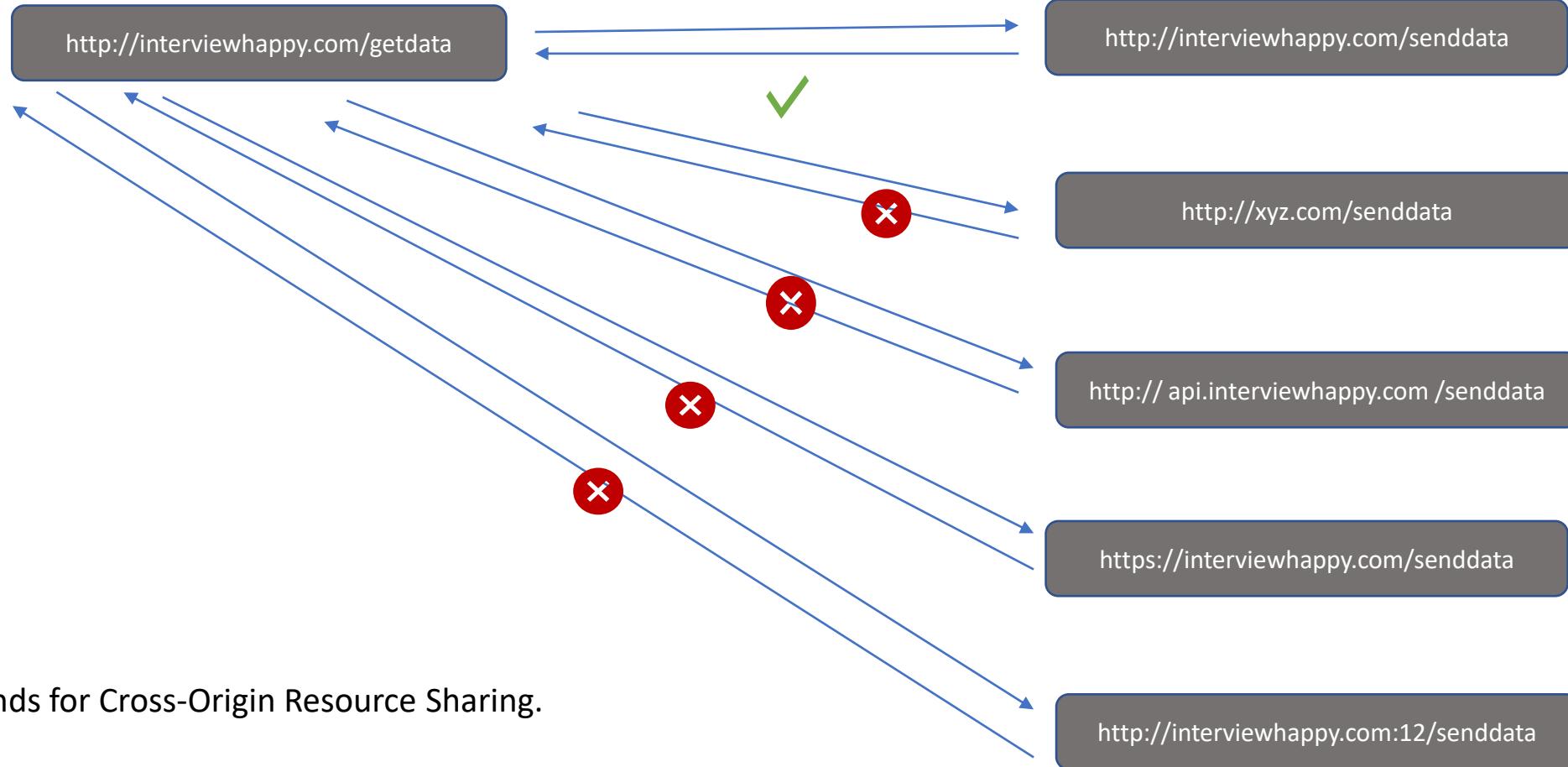
```

 1  {
 2   "iisSettings": {
 3     "windowsAuthentication": false,
 4     "anonymousAuthentication": true,
 5     "iisExpress": {
 6       "applicationUrl": "http://localhost:50644",
 7       "sslPort": 44309
 8     }
 9   },
10   "$schema": "http://json.schemastore.org/launchsettings.json",
11   "profiles": {
12     "IIS Express": {
13       "commandName": "IISExpress",
14       "launchBrowser": true,
15       "launchUrl": "api/values",
16       "environmentVariables": {
17         "ASPNETCORE_ENVIRONMENT": "Development"
18       }
19     },
20     "HeaderHelper": {
21       "commandName": "Project",
22       "launchBrowser": true,
23       "launchUrl": "api/values",
24       "environmentVariables": {
25         "ASPNETCORE_ENVIRONMENT": "Development"
26       },
27       "applicationUrl": "https://localhost:5001;http://localhost:5000"
28     },
29     "Docker": {
30       "commandName": "Docker",
31       "launchBrowser": true,
32       "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}/api/values",
33       "httpPort": 50645,
34       "useSSL": true,
35       "sslPort": 44310
36     }
37   }
38 }

```



- ❖ Appsettings.json (Default) (Mostly Used)
- ❖ Azure Key Vault (Application is hosted on Azure)
- ❖ Environment variables
- ❖ In-memory .NET objects
- ❖ Command Line Arguments
- ❖ Custom Providers



- ❖ CORS stands for Cross-Origin Resource Sharing.
- ❖ CORS is a security feature implemented in web browsers that restricts web pages or scripts from making requests to a different domain than the one that served the web page.

✖ Access to XMLHttpRequest at 'http://localhost:5000/global_config' [step1:1](#) from origin '<http://localhost:8080>' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

- ❖ CORS stands for Cross-Origin Resource Sharing.
- ❖ CORS is a security feature implemented in web browsers that restricts web pages or scripts from making requests to a different domain than the one that served the web page.

❖ Why CORS is required?

Because of security reason. So that you can make your api secure by default. Otherwise, any external website can try to access your data.

❖ How to fix CORS error?

1. Add Microsoft.AspNetCore.Cors nuget package to your project.
2. In startup class, edit the Configure and ConfigureServices method.

```
// This method gets called by the runtime. Use this method to add service
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
    services.AddControllers();
}

// This method gets called by the runtime. Use this method to configure the
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();

    // global cors policy
    app.UseCors(x => x
        .AllowAnyMethod()
        .AllowAnyHeader()
        .SetIsOriginAllowed(origin => true) // allow any origin
        .AllowCredentials()); // allow credentials

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(x => x.MapControllers());
}
```

In-Memory Caching

1. It's the normal way of caching.
In this cache is stored in the memory of a **single server** which is hosting the application.

2. It can be implemented with the `IMemoryCache` Interface in ASP.NET Core.

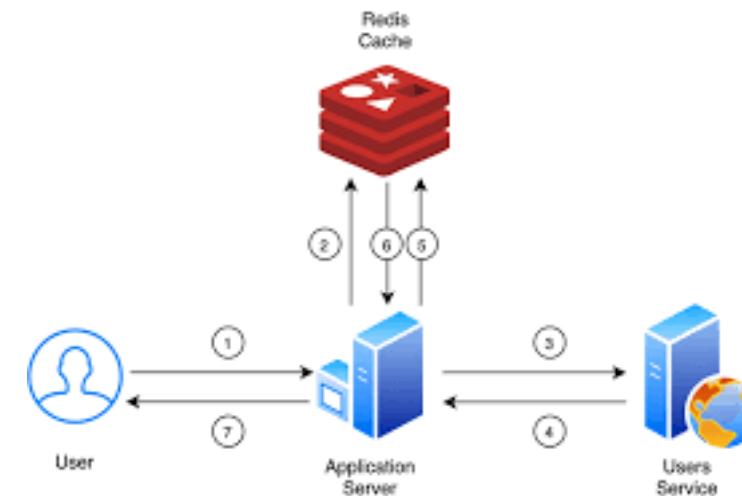
- ❖ Redis is an open-source, highly replicated, performant, non-relational kind of database and caching server.
- ❖ When to use which caching?

In normal cases where the application size is small, use in-memory cache.
But where application is very big or it's a microservices based architecture, then use distributed caching.

Distributed Caching

Distributed caching is when you want to handle caching outside of your application. A different server is used for store cached data.

It can be implemented with the help of Redis Cache.



- ❖ Error handling for development and other environments can be set in **CONFIGURE** method of **Startup.cs** class.

```
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
}
```

- ❖ The **IsDevelopment()** method will check the **ASPNETCORE_ENVIRONMENT** value in **LaunchSettings.json** file.

```
"Example": {
    "commandName": "Project",
    "dotnetRunMessages": "true",
    "launchBrowser": true,
    "applicationUrl": "http://localhost:5000",
    "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
    }
}
```

- ❖ Razor pages follows a page-centric development model just like ASP.NET web forms.

Chapter 28 : SOLID Principles

Q234. What are **SOLID Principles**? What is the difference between SOLID Principles and Design Patterns? **V Imp**

Q235. What is **Single Responsibility Principle**? **V Imp**

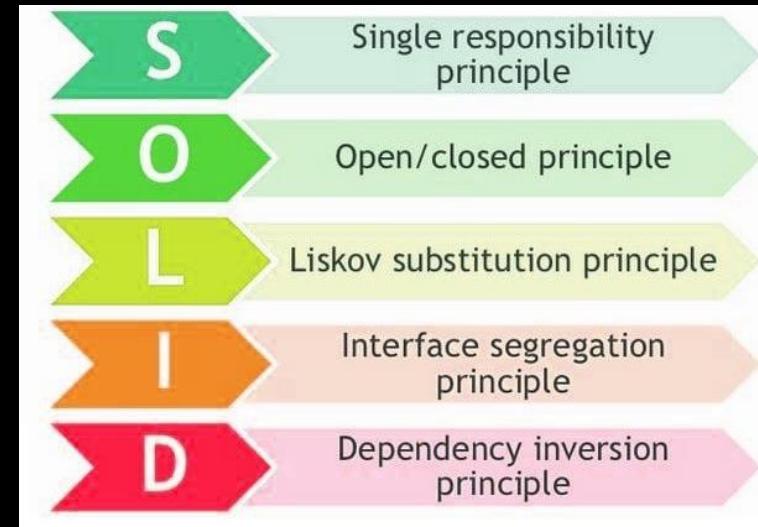
Q236. What is **Open-closed Principle**?

Q237. What is **Liskov Substitution Principle**? **V Imp**

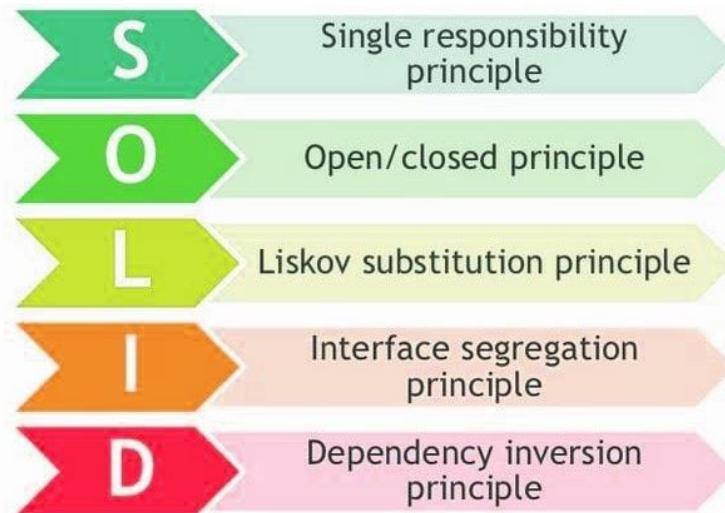
Q238. What is **Interface Segregation Principle**?

Q239. What is **Dependency Inversion Principle**?

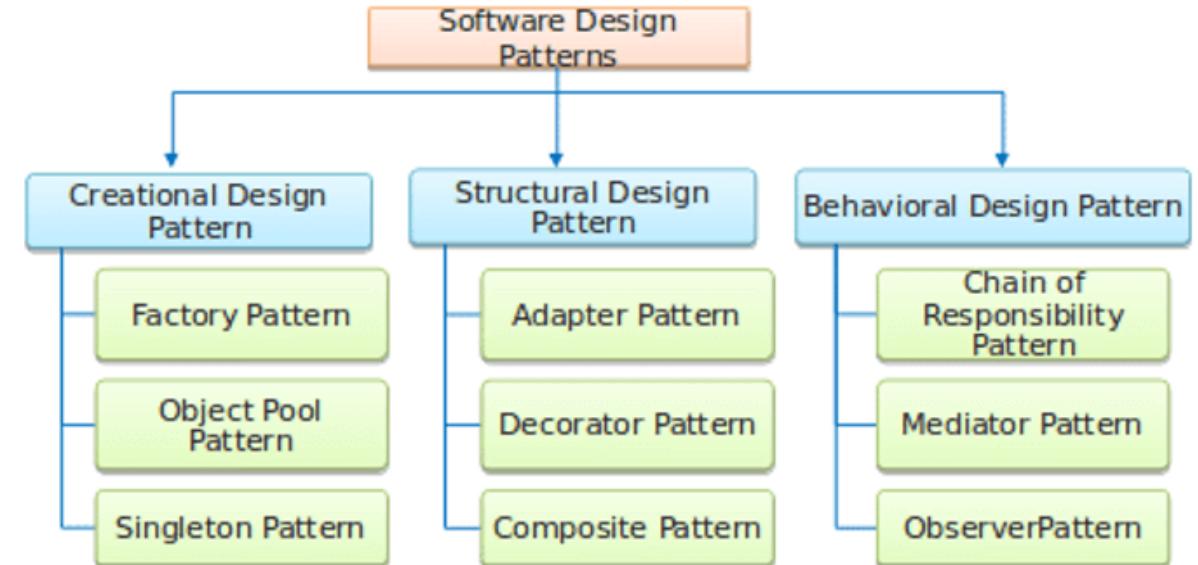
Q240. What is **DRY principle**?



- ❖ SOLID principles are a set of principles, which must be followed to develop flexible, maintainable, and scalable software systems.



- ❖ Design patterns are concrete and solve a particular kind of problem in software's.



- ❖ SOLID principles aren't concrete - rather abstract.

- ❖ Design patterns are concrete and solve a particular kind of problem in a specific and fixed manner.

- ❖ Single Responsibility Principle (SRP) states that a class should have only **one responsibility**.
- ❖ Or a class should have only **one reason** to change.

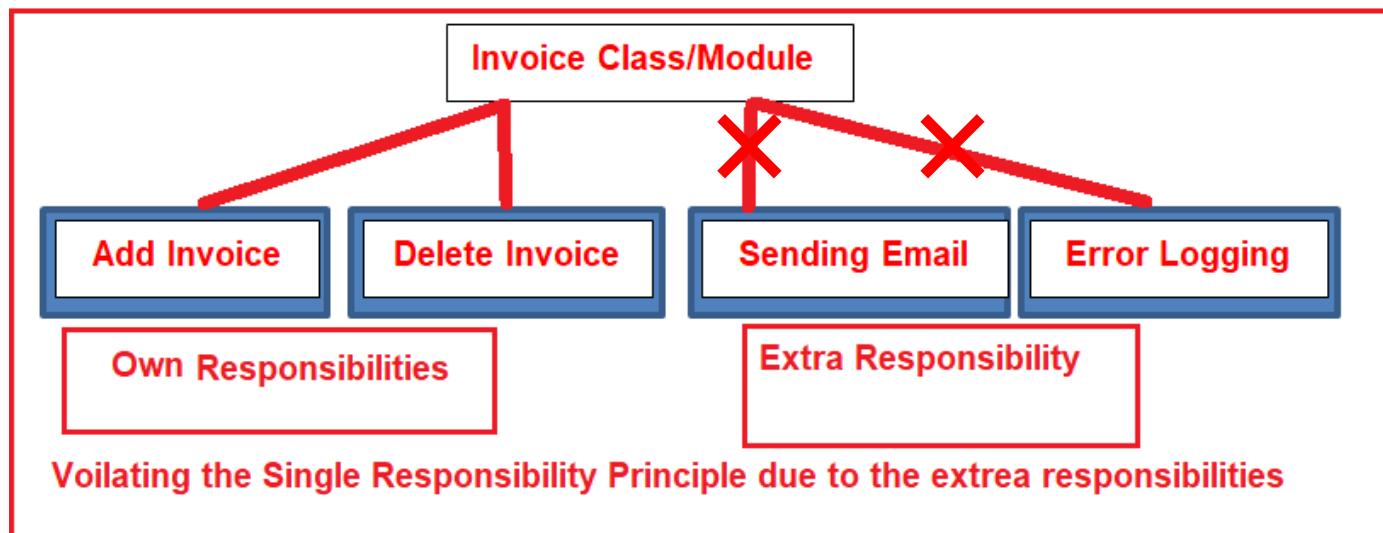
```
//Violating SRP, because the class
//has extra responsibility

public class Employee
{
    //Own responsibility
    public int CalculateSalary()
    {
        return 100000;
    }

    //Own responsibility
    public string GetDepartment()
    {
        return "IT";
    }

    //Extra responsibility
    public void Save()
    {
        //Save employee to the database
    }
}
```

- ❖ Single Responsibility Principle (SRP) states that a class should have only **one responsibility**.
- ❖ Or a class should have only **one reason** to change.
- ❖ When a class has only one responsibility, it becomes easier to change and test. If a class has multiple responsibilities, changing one responsibility may impact others and more testing efforts will be required then.



```
//Following SRP
public class Employee
{
    public int CalculateSalary()
    {
        return 100000;
    }
    public string GetDepartment()
    {
        return "IT";
    }
}

public class EmployeeRepository
{
    public void Save(Employee employee)
    {
        //Save employee to the database
    }
}
```

- ❖ Open-Closed Principle (OCP) states that software entities(classes, modules) should be open for **extension**, but closed for **modification**.

Inserting one more else if{} will violate the open closed principle because you are modifying the class rather than extending it.

```
public class Account
{
    public string Name { get; set; }

    public string Address { get; set; }

    public double Balance { get; set; }

    public double CalculateInterest(string accountType)
    {
        if(accountType == "Saving")
        {
            return Balance * 0.3;
        }
        else
        {
            return Balance * 0.5;
        }
    }
}
```

- ❖ Open-Closed Principle (OCP) states that software entities(classes, modules) should be open for **extension**, but closed for **modification**.
- ❖ SRP is the **prerequisite** for OCP.

```
public class Account
{
    public string Name { get; set; }
    public string Address { get; set; }
    public double Balance { get; set; }
}
```

```
interface IAccount
{
    double CalculateInterest(Account account);
}
```

- ❖ The benefit is **simple testing** is required to test individual classes, but if you will keep on adding and modifying in one class. Then even for the smallest modification, the whole class needs to be tested.

```
public class SavingAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.3;
    }
}

public class OtherAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.5;
    }
}

public class CurrentAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.7;
    }
}
```

- ❖ The Liskov Substitution Principle (LSP) states that an object of a child class must be able to **replace** an object of the parent class without **breaking** the application.

```
static void Main(string[] args)
{
    Employee employee = new Employee();

    PermanentEmployee pEmployee = new PermanentEmployee();

    ContractualEmployee cEmployee = new ContractualEmployee();

    Console.WriteLine(employee.CalculateSalary());
    //100000
    Console.WriteLine(employee.CalculateBonus());
    //10000

    Console.WriteLine(pEmployee.CalculateSalary());
    //200000
    Console.WriteLine(pEmployee.CalculateBonus());
    //10000
    Console.WriteLine(cEmployee.CalculateSalary());
    //150000
    Console.WriteLine(cEmployee.CalculateBonus());
    //Exception: The method or operation is not implemented
}
```

```
//Base/ Parent/ Superclass
public class Employee
{
    public virtual int CalculateSalary()
    {
        return 100000;
    }
    public virtual int CalculateBonus()
    {
        return 10000;
    }
}

//Derived/ Child/ Subclass
public class PermanentEmployee : Employee
{
    public override int CalculateSalary()
    {
        return 200000;
    }
}

public class ContractualEmployee : Employee
{
    public override int CalculateSalary()
    {
        return 150000;
    }
    public override int CalculateBonus()
    {
        throw new NotImplementedException();
    }
}
```

- ❖ All the base class methods must be applicable for the derived class.

[back to chapter index](#)

- ❖ The Interface Segregation (ISP) states that a class should not be forced to implement interfaces that it does not use.

```
public interface IVehicle
{
    void Drive();

    void Fly();
}
```

```
public class FlyingCar : IVehicle
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

    public void Fly()
    {
        Console.WriteLine("Fly car");
    }
}
```

```
public class Car : IVehicle
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

    public void Fly()
    {
        throw new NotImplementedException();
    }
}
```

- ❖ The Interface Segregation (ISP) states that a class should not be forced to implement interfaces that it does not use.
- ❖ It is better to have multiple smaller interfaces than larger interfaces.

```
public interface IDrive
{
    void Drive();
}

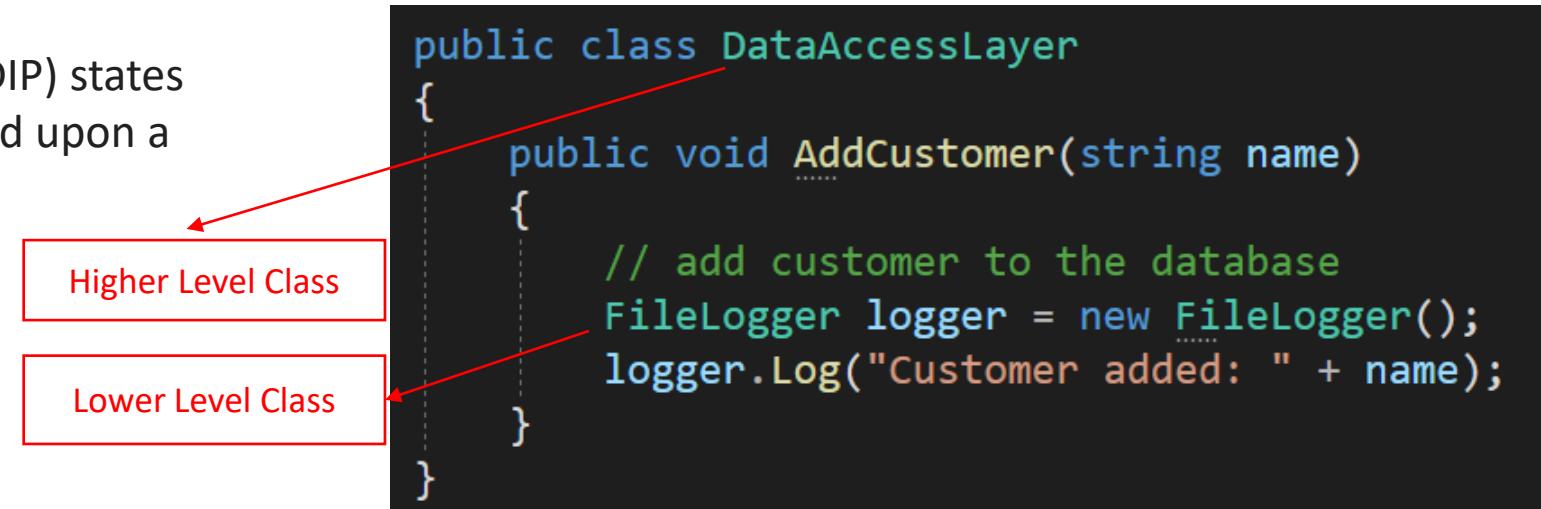
public interface IFly
{
    void Fly();
}
```

```
public class Car : IDrive
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }
}
```

```
public class FlyingCar : IDrive, IFly
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

    public void Fly()
    {
        Console.WriteLine("Fly car");
    }
}
```

- ❖ The Dependency Inversion Principle (DIP) states that a **high-level class** must not depend upon a **lower level class**.



```
public class FileLogger
{
    public void Log(string message)
    {
        // write message to a log file
    }
}
```

- ❖ The Dependency Inversion Principle (DIP) states that a **high-level class** must not depend upon a **lower level class**.

```
public interface ILogger
{
    void Log(string message);
}

public class FileLogger : ILogger
{
    public void Log(string message)
    {
        // write message to a log file
    }
}
```

```
public class DataAccessLayer
{
    private ILogger logger;

    public DataAccessLayer(ILogger logger)
    {
        this.logger = logger;
    }

    public void AddCustomer(string name)
    {
        // add customer to the database
        logger.Log("Customer added: " + name);
    }
}
```

- ❖ DRY stands for DON'T REPEAT YOURSELF.
- ❖ Don't write the same functionality multiple time.
- ❖ Instead write code once and then use it at multiple places using inheritance.

Chapter 29 : Design Patterns

Q241. What are **Design Patterns** and what problem they solve? **V Imp**

Q242. What are the types of Design Patterns? **V Imp**

Q243. What are **Creational** Design Patterns?

Q244. What are **Structural** Design Patterns?

Q245. What are **Behavioral** Design Patterns?

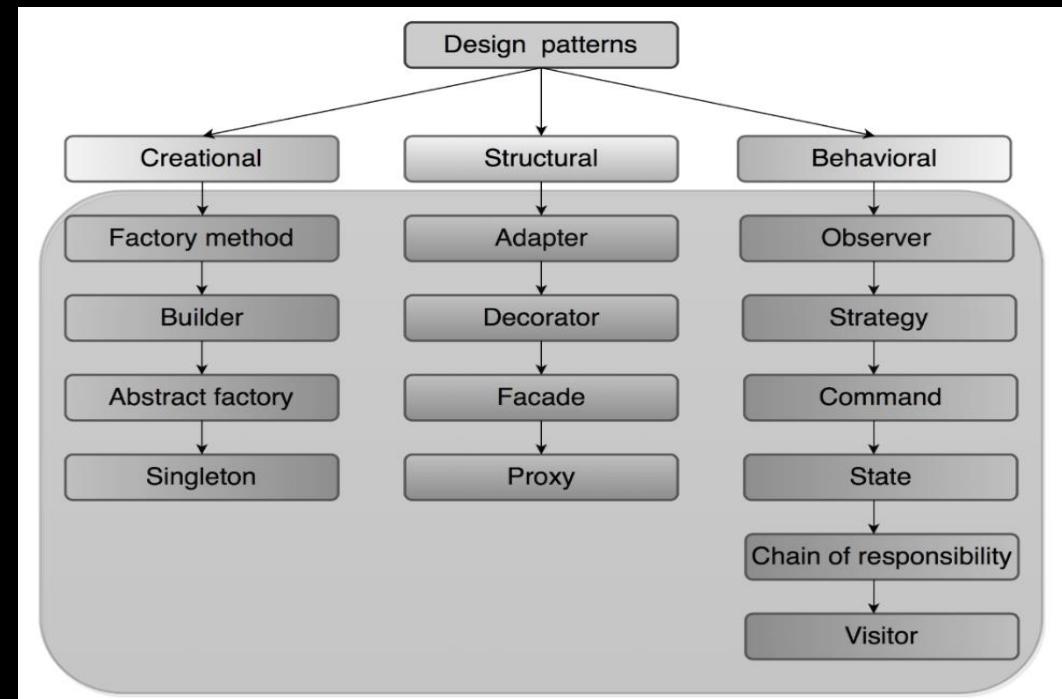
Q246. What is **Singleton** Design Pattern? **V Imp**

Q247. How to make singleton pattern **thread safe**? **V Imp**

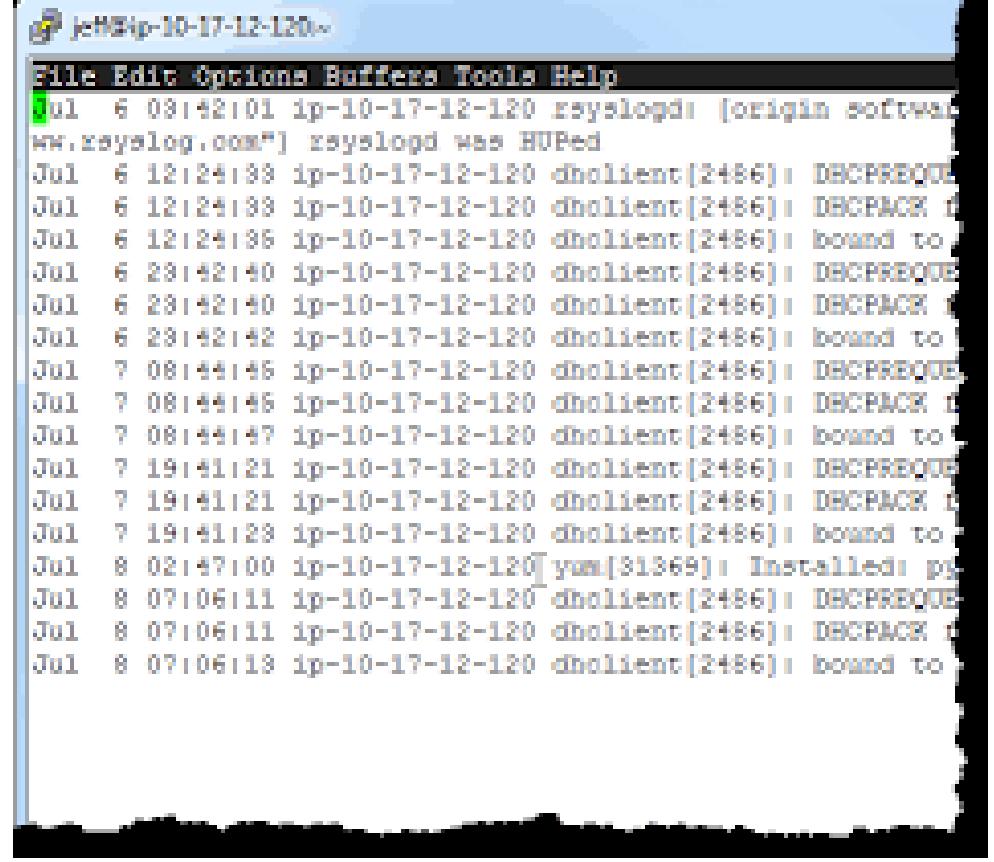
Q248. What is **Factory** pattern? Why to use factory pattern?

Q249. How to **implement** Factory method pattern?

Q250. What is **Abstract Factory** pattern?

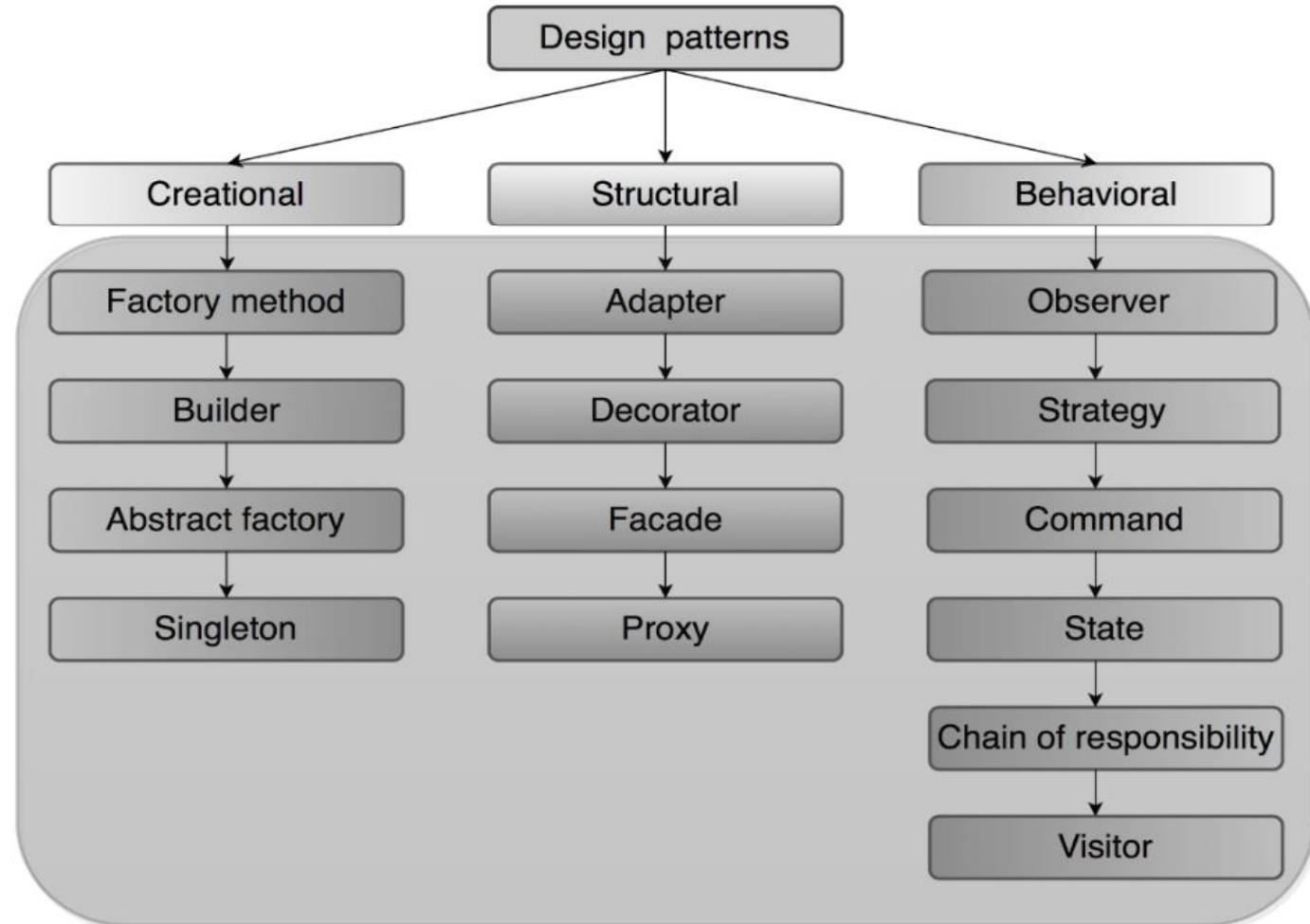


- ❖ Design patterns are **reusable solutions** for common problems in software design.



The screenshot shows a terminal window with a blue header bar. The title bar contains the text 'jet@ip-10-17-12-120:~'. The window has a standard window frame with a close button in the top right corner. The main area of the terminal is a black text area with white text, displaying a log of DHCP client activity. The log entries are as follows:

```
File Edit Options Buffers Tools Help
Jul 6 09:42:01 ip-10-17-12-120 rsyslogd: [origin software="www.rsyslog.com"] rsyslogd was HUPed
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 6 12:24:36 ip-10-17-12-120 dhclient[2486]: bound to
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 6 23:42:42 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 7 08:44:47 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 7 19:41:23 ip-10-17-12-120 dhclient[2486]: bound to
Jul 8 02:47:00 ip-10-17-12-120 yum[31369]: Installed: ps
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 8 07:06:13 ip-10-17-12-120 dhclient[2486]: bound to
```

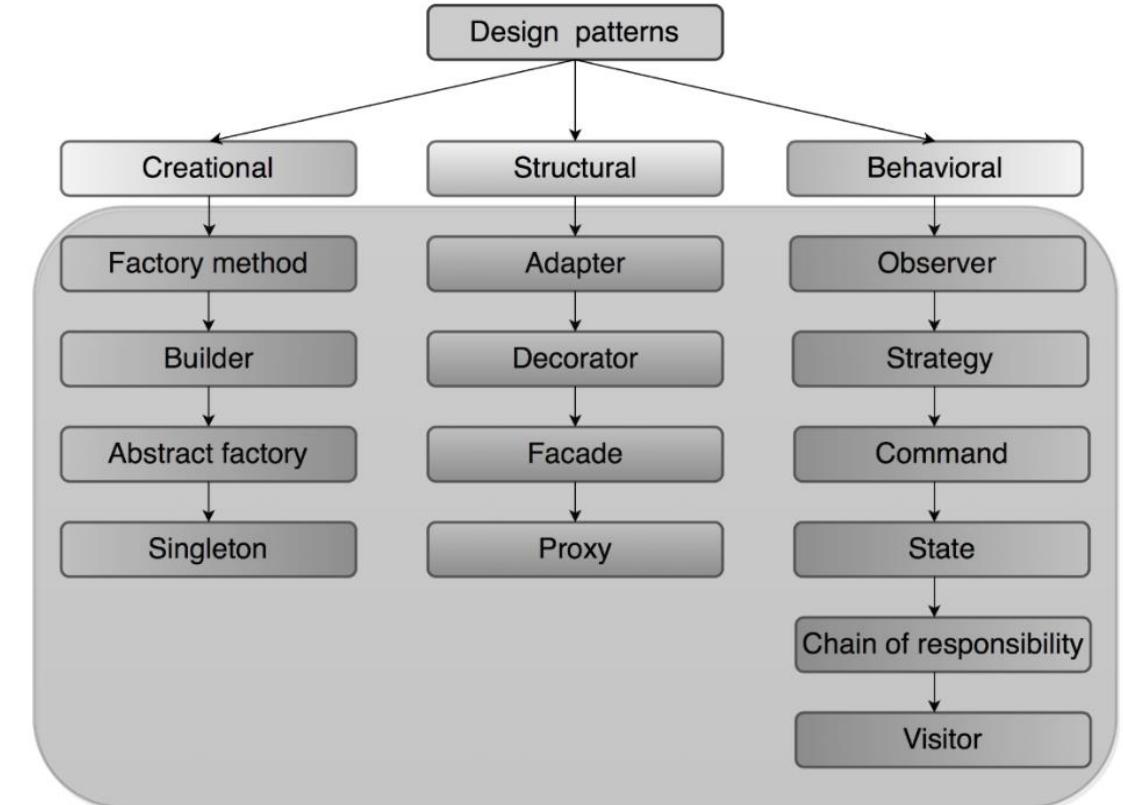


- ❖ Creational design patterns are design patterns that deal with **object creation** mechanisms.

- ❖ Why direct object creation is a problem?

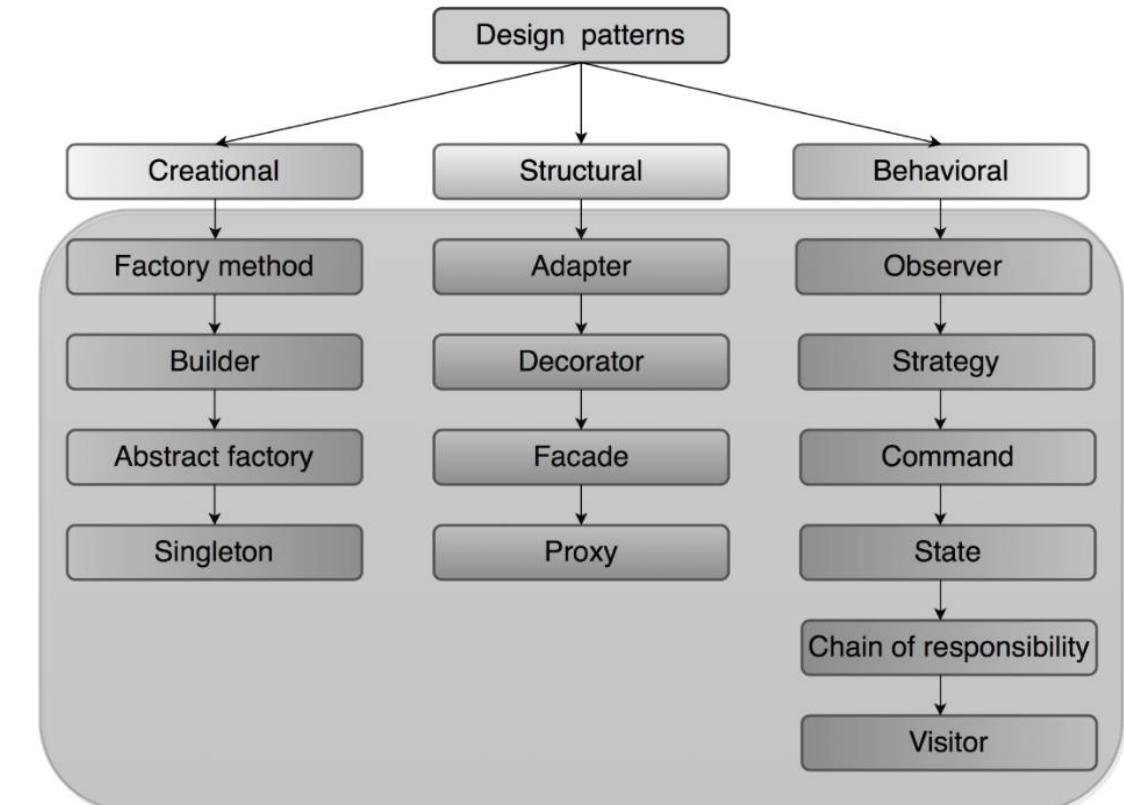
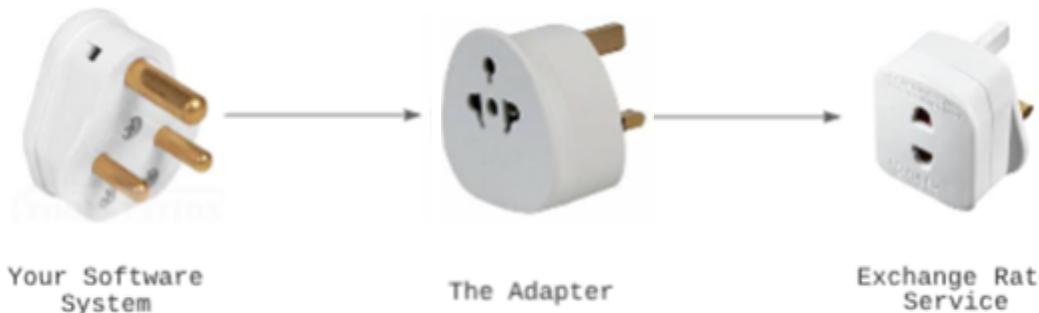
```
public Class Pet
{
    Cat b = new Cat();
    b.CountLegs();
}
```

- ❖ The reason is in a large/enterprise application, if you want to change this Class from Cat to Dog, then you have to change it everywhere wherever you have used it and then it will be a testing issue. You have to retest all these classes like Pet class.

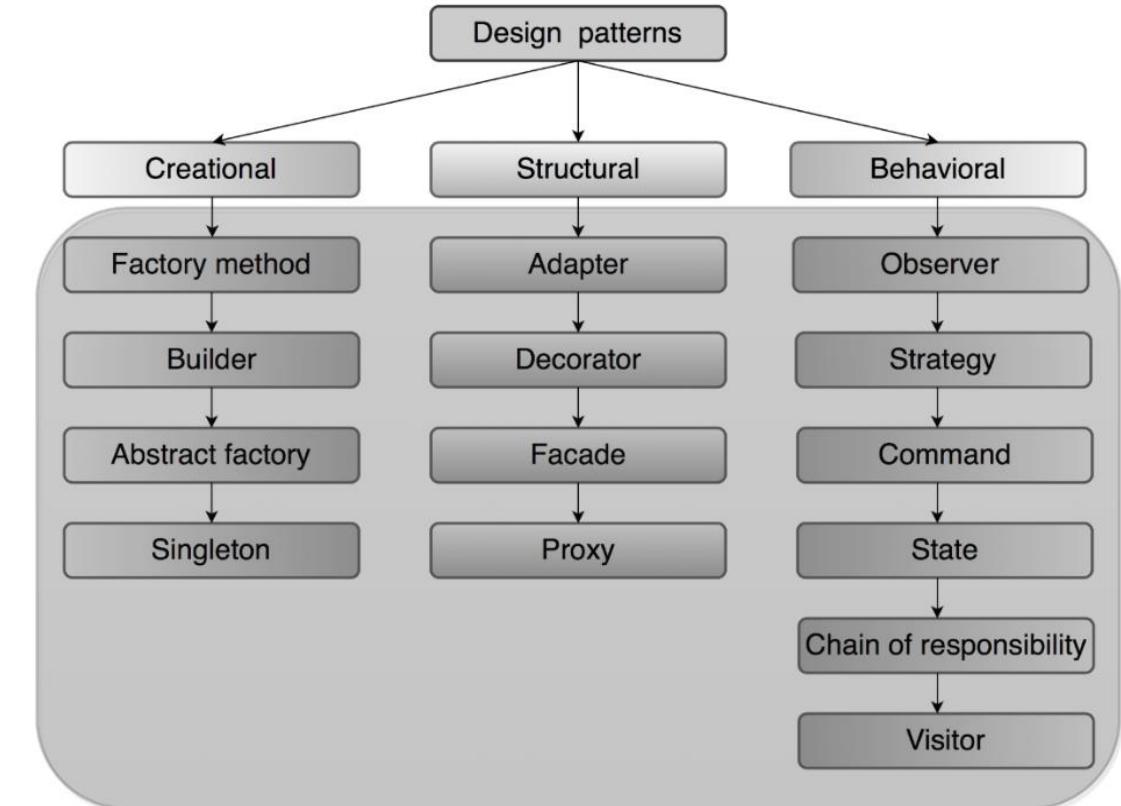
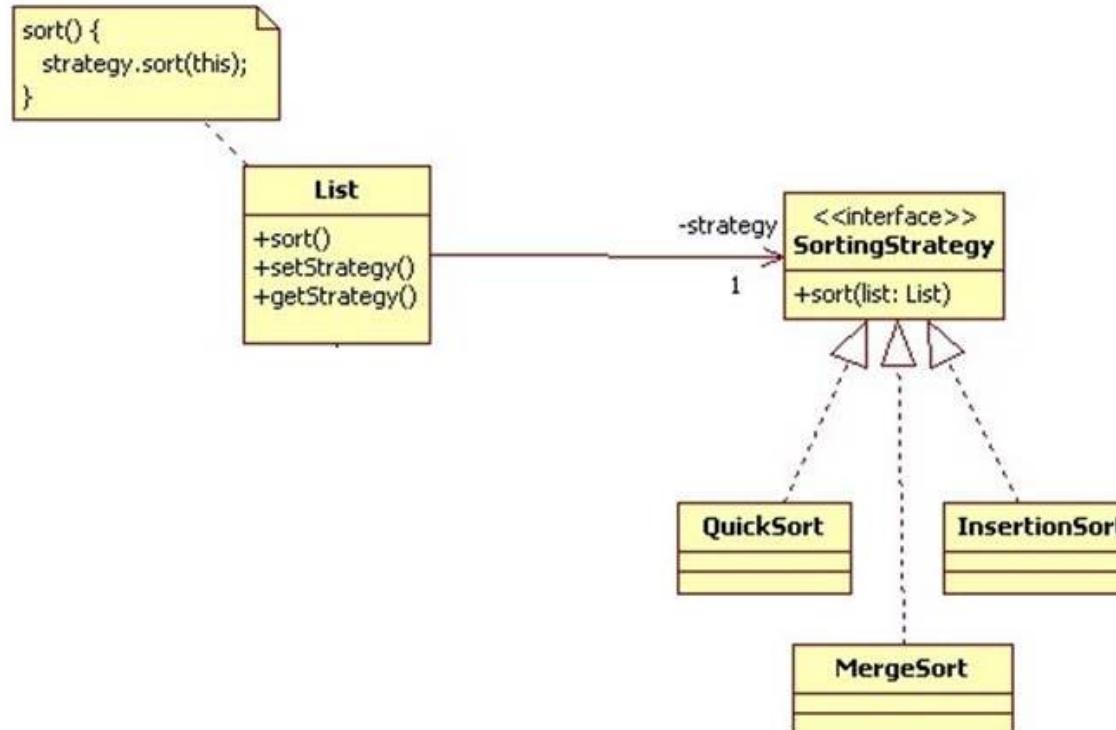


- ❖ Structural Design Pattern is used to manage the structure of classes and interfaces to manage the **relationship** between the classes.

- ❖ Adapter pattern



- ❖ Behavioral design patterns defines the way to **communicate** between classes and object.
- ❖ For example, in strategy pattern we can choose the best strategy at run time for doing any operation .



- ❖ The Singleton design pattern is a creational design pattern that ensures a class has only **one instance** of it.

```
public class NonSingleton
{
    public void Method()
    {
        //logic
    }
}

static void Main(string[] args)
{
    NonSingleton obj = new NonSingleton();
    obj.Method();
}

static void Main(string[] args)
{
    Singleton obj = Singleton.GetInstance();
    obj.Method();
}
```

1. Declare class as sealed to prevent any further inheritance

2. Declare a static instance of this class.

3. Create a private constructor so that no one can create the object of this class.

4. Create a method which will check whether the instance is null, if yes then it will create the new instance and if not null then it will return the same instance.

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    public static Singleton GetInstance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

    public void Method()
    {
        //logic..
    }
}
```

❖ Normal Singleton Pattern

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    public static Singleton GetInstance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

    public void Method()
    {
        //logic..
    }
}
```



```
//Thread Safe Singleton Pattern

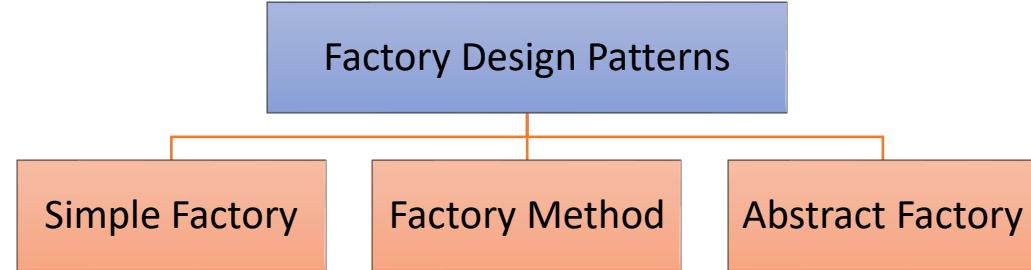
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    private static readonly object padlock = new object();

    public static Singleton GetInstance()
    {
        lock (padlock)
        {
            if (instance == null)
            {
                instance = new Singleton();
            }
            return instance;
        }
    }
}
```

- ❖ The Factory Pattern is a **creational** design pattern which manages object creation.



```

// Concrete Classes
class Java
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}

class DotNet
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
  
```

```

// Client code
class Program
{
    static void Main(string[] args)
    {
        // Creating Java object
        Java java = new Java();
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        DotNet dotnet = new DotNet();
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
  
```

Two red callout boxes with arrows point to the client code:

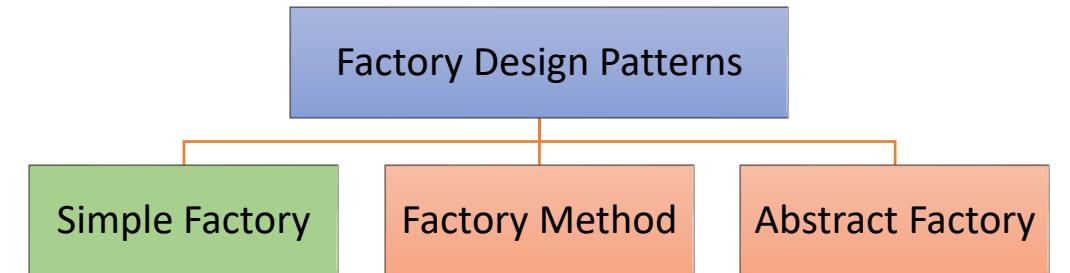
- 1. Lots of **new** keywords
- 2. Client will know all the concrete classes name

- ❖ Factory Pattern provides an **interface** or abstract class for creating objects, but lets the subclasses decide which class to instantiate.
- ❖ Or In simple factory pattern, one factory class is responsible for creating all the objects for its derived or sub class.

```
// Interface
interface ITech
{
    string GetFramework();
}

// Concrete Classes
class Java : ITech
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}
class DotNet : ITech
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
```

```
// Simple Factory
class TechFactory
{
    public ITech CreateTech(string techType)
    {
        if (techType.Equals("Java"))
        {
            return new Java();
        }
        else if (techType.Equals("DotNet"))
        {
            return new DotNet();
        }
        else
        {
            throw new Exception("Invalid");
        }
    }
}
```



```
// Client code
class Program
{
    static void Main(string[] args)
    {
        TechFactory factory = new TechFactory();

        // Creating Java object
        ITech java = factory.CreateTech("Java");
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        ITech dotnet = factory.CreateTech("DotNet");
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
```

- ❖ Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.

```
// Interface
interface ITech
{
    string GetFramework();
}

// Concrete Classes
class Java : ITech
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}

class DotNet : ITech
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
```

```
// Simple Factory
class TechFactory
{
    public ITech CreateTech(string techType)
    {
        if (techType.Equals("Java"))
        {
            return new Java();
        }
        else if (techType.Equals("DotNet"))
        {
            return new DotNet();
        }
        else
        {
            throw new Exception("Invalid");
        }
    }
}
```

```
// Client code
class Program
{
    static void Main(string[] args)
    {
        TechFactory factory = new TechFactory();

        // Creating Java object
        ITech java = factory.CreateTech("Java");
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        ITech dotnet = factory.CreateTech("DotNet");
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
```

- ❖ Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.

```
// Product
interface ITech
{
    void GetFramework();
}

// Concrete Product
class Java : ITech
{
    public void GetFramework()
    {
        Console.WriteLine("Spring");
    }
}

// Concrete Product
class DotNet : ITech
{
    public void GetFramework()
    {
        Console.WriteLine("Core");
    }
}
```

```
abstract class Creator
{
    // The Factory Method
    public abstract ITech CreateTech();
}

// Concrete Creator
class JavaCreator : Creator
{
    // Implementation of Factory Method
    public override ITech CreateTech()
    {
        return new Java();
    }
}

// Concrete Creator
class DotNetCreator : Creator
{
    // Implementation of Factory Method
    public override ITech CreateTech()
    {
        return new DotNet();
    }
}
```

```
static void Main(string[] args)
{
    // Create a JavaCreator object
    Creator javaCreator = new JavaCreator();

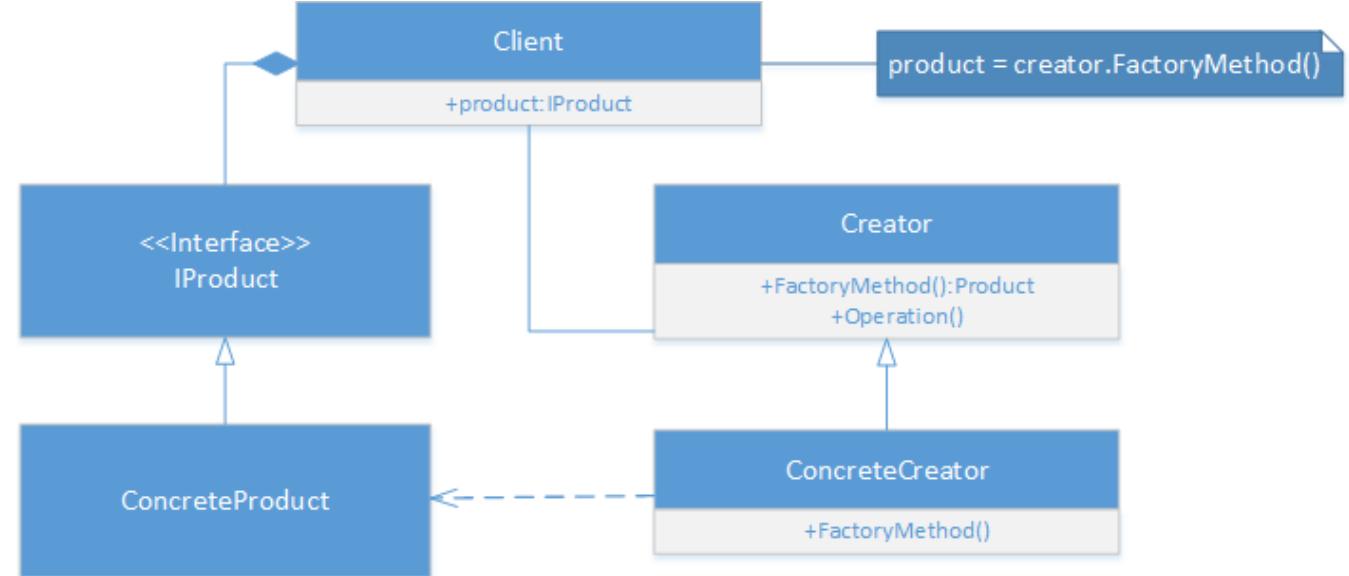
    // Call Factory Method to
    // create a JavaTech object
    ITech javaTech = javaCreator.CreateTech();
    javaTech.GetFramework();

    // Create a DotNetCreator object
    Creator dotNetCreator = new DotNetCreator();

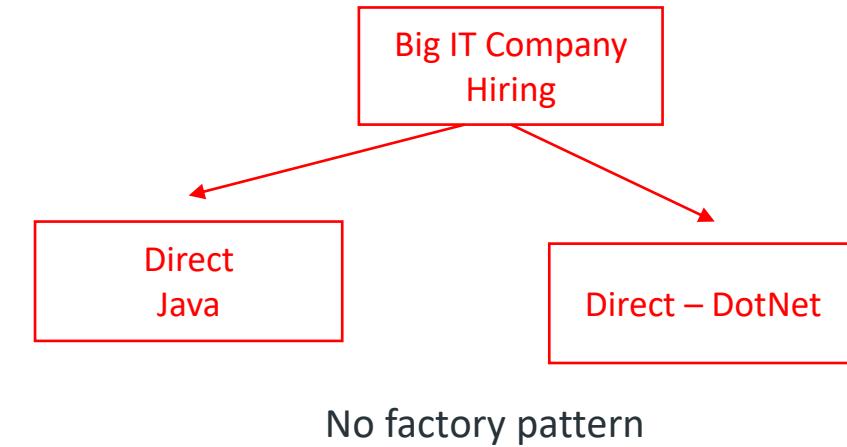
    // Call Factory Method to
    // create a DotNetTech object
    ITech dotNetTech = dotNetCreator.CreateTech();
    dotNetTech.GetFramework();

    Console.ReadLine();
}
//Output: Spring
//Output: Core
```

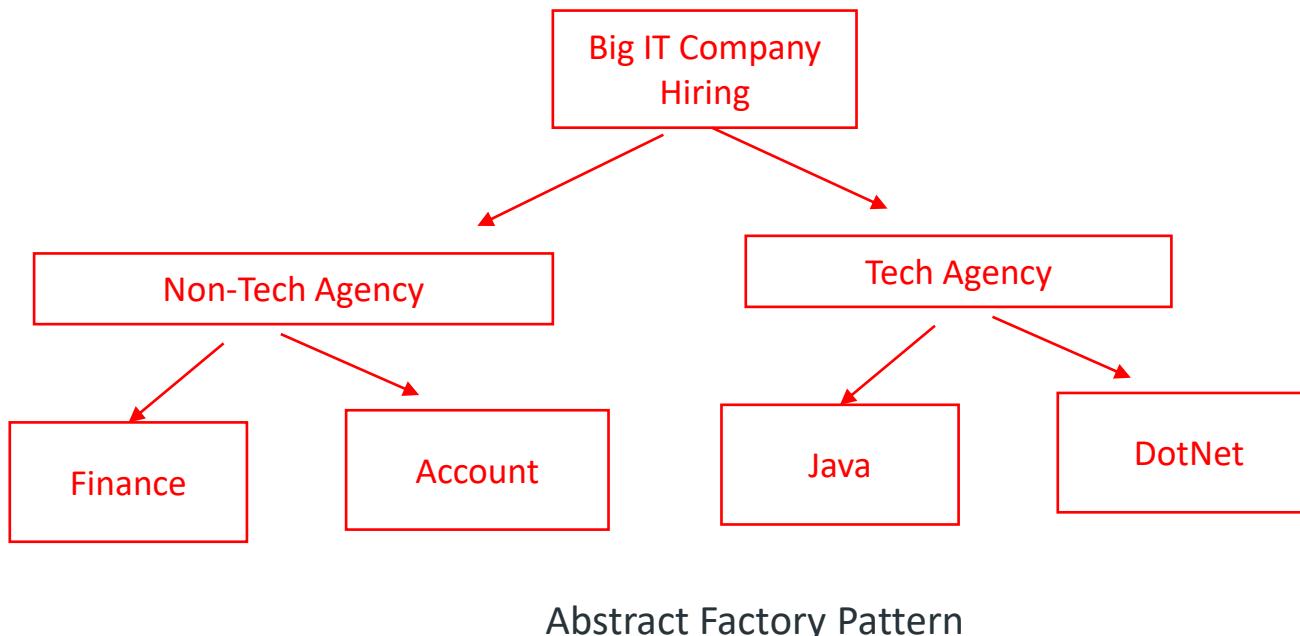
- ❖ Things to remember about Factory(simple factory and factory method) design patterns.
1. The Factory Pattern is a **creational** design pattern which manages object creation.
 2. Factory Pattern provides an **interface** or abstract class for creating objects, but lets the subclasses decide which class to instantiate.
 3. Or In simple factory pattern, one factory class is responsible for creating all the objects for its derived or sub class.
 4. Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.



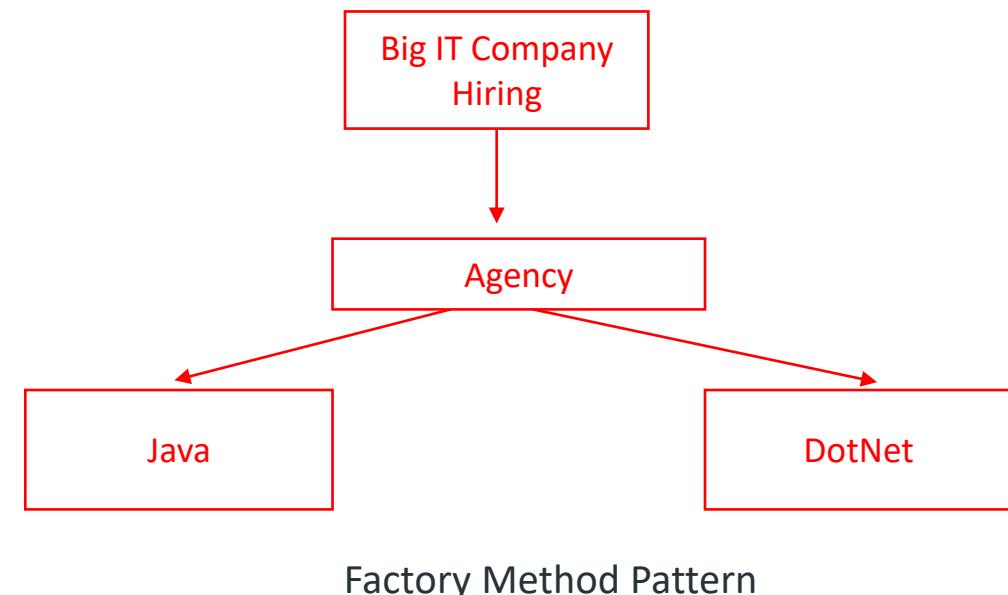
- ❖ The Abstract Factory pattern is a creational design pattern that provides an interface for creating related families of objects without specifying their concrete classes.
- ❖ Abstract Factory pattern can be thought of as a "factory of factories".



No factory pattern



Abstract Factory Pattern



Factory Method Pattern



All the best for your interviews

Never ever give up



TOP

500

.NET

INTERVIEW

QUESTIONS

PART - II

HAPPY RAWAT

PREFACE

ABOUT THE BOOK

Part II - This book contains 250 more interview questions.

ABOUT THE AUTHOR

Happy Rawat has around 15 years of experience in software development. He helps candidates in clearing technical interview in tech companies.

Topic	Part II (Number of Questions)
OOPS/ C#	38
.NET FRAMEWORK	27
SQL	25
ASP.NET MVC	52
ASP.NET WEBFORMS	35
ADO.NET & EF	10
WEB API	37
.NET CORE	17
SOLID PRINCIPLES	0
DESIGN PATTERNS	9
TOTAL	250

OOPS

1. WHAT IS THE DIFFERENCE BETWEEN A CLASS AND A STRUCTURE?

- CLASS: User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.
 - STRUCTURE: A structure is basically a user-defined collection of variables which are of different data types.
-

2. WHAT IS OPERATOR OVERLOADING?

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it.

3. CAN YOU CALL THE BASE CLASS METHOD WITHOUT CREATING AN INSTANCE?

- Yes, you can call the base class without instantiating it if:
 - It is a static method.
 - The base class is inherited by some other subclass.
-

4. WHAT ARE VIRTUAL FUNCTIONS AND PURE VIRTUAL FUNCTIONS?

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

Pure virtual functions or abstract functions are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

5. WHICH OOPS CONCEPT EXPOSES ONLY THE NECESSARY INFORMATION TO THE CALLING FUNCTIONS?

Encapsulation

6. IS IT ALWAYS NECESSARY TO CREATE OBJECTS FROM CLASS?

No, it is possible to call a base class method if it is defined as a static method.

7. WHAT IS EARLY AND LATE BINDING?

Early binding refers to the task of values to variables during plan time, whereas late Binding refers to the assignment of values to variables during run time.

8. WHICH OOPS CONCEPT IS USED AS A REUSE MECHANISM?

Inheritance is the OOPS concept that can be used as a reuse mechanism.

9. IN TRY BLOCK IF WE ADD RETURN STATEMENT WHETHER FINALLY BLOCK IS EXECUTED?

Yes. Finally block will still be executed in presence of return statement in try block.

10. WHAT YOU MEAN BY INNER EXCEPTION?

Inner exception is a property of exception class which will give you a brief insight of the exception i.e, parent exception and child exception details.

11. LIST OUT SOME OF THE EXCEPTIONS?

Below are some of the exceptions -

- NullReferenceException
 - ArgumentNullException
 - DivideByZeroException
 - IndexOutOfRangeException
 - InvalidOperationException
 - StackOverflowException etc.
-

12. WHAT IS THE DIFFERENCE BETWEEN STACK AND QUEUE COLLECTIONS?

A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the **top**. A stack follows the **LIFO** (Last In First

Out) principle, i.e., the element inserted at the last is the first element to come out.

A queue is a linear data structure in which elements can be inserted only from one side of the list called **rear**, and the elements can be deleted only from the other side called the **front**. The queue data structure follows the **FIFO** (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list.

13. EXPLAIN JAGGED ARRAYS?

If the elements of an array is an array then it's called as jagged array. The elements can be of different sizes and dimensions.

14. EXPLAIN MULTIDIMENSIONAL ARRAYS?

A multi-dimensional array in C# is an array that contains more than one rows to store the data. The multidimensional array can be declared by adding commas in the square brackets.

15. EXPLAIN INDEXERS?

Indexers are used for allowing the classes to be indexed like arrays. Indexers will resemble the property structure but only difference is indexer's accessors will take parameters. For example,

```
class MyCollection<T>
{
    private T[] myArr = new T[100];
    public T this[int t]
```

```
{  
get  
{  
    return myArr[t];  
}  
set  
{  
    myArr[t] = value;  
}  
}  
}
```

16. WHAT IS THE DIFFERENCE BETWEEN METHODS – “SYSTEM.ARRAY.CLONE()” AND “SYSTEM.ARRAY.COPYTO()”?

- “CopyTo()” method can be used to copy the elements of one array to other.
 - “Clone()” method is used to create a new array to contain all the elements which are in the original array.
-

17. WHAT ARE VALUE TYPE AND REFERENCE TYPES?

Value types stored in a stack.

Here are the value types:

- Decimal, int, byte, enum, double, long, float

Reference types are stored in a heap.

Below are the list of reference types -

- Class, string, interface, object

18.CAN WE OVERRIDE PRIVATE VIRTUAL METHOD?

No. We can't override private virtual methods as it is not accessible outside the class.

19.EXPLAIN CIRCULAR REFERENCE?

This is a situation where in, multiple resources are dependent on each other and this causes a lock condition and this makes the resource to be unused.

20.EXPLAIN OBJECT POOL?

Object pool is used to track the objects which are being used in the code. So object pool reduces the object creation overhead.

21.WHAT ARE THE TYPES OF DELEGATES?

Below are the uses of delegates -

- Single Delegate
 - Multicast Delegate
 - Generic Delegate
-

22.WHAT ARE THE THREE TYPES OF GENERIC DELEGATES?

Below are the three types of generic delegates -

- Func

- Action
 - Predicate
-

23. WHAT ARE THE USES OF DELEGATES?

Below are the list of uses of delegates -

- Callback Mechanism
 - Asynchronous Processing
 - Abstract and Encapsulate method
 - Multicasting
-

24. CAN WE USE DELEGATES FOR ASYNCHRONOUS METHOD CALLS?

Yes. We can use delegates for asynchronous method calls.

25. WHAT IS AN ESCAPE SEQUENCE? NAME SOME STRING ESCAPE SEQUENCES IN C#.

An Escape sequence is denoted by a backslash (\). The backslash indicates that the character that follows it should be interpreted literally or it is a special character. An escape sequence is considered as a single character.

26. WHAT ARE REGULAR EXPRESSIONS?

Regular expression is a template to match a set of input. The pattern can consist of operators, constructs or character literals. Regex is used for string parsing and replacing the character string.

27.WHY TO USE LOCK STATEMENT?

Lock will make sure one thread will not intercept the other thread which is running the part of code. So lock statement will make the thread wait, block till the object is being released.

28.WHAT IS “EXTERN” KEYWORD?

The **extern** modifier is used to declare a method that is implemented externally. A common use of the **extern** modifier is with the **DllImport** attribute when you are using Interop services to call into unmanaged code.

```
[DllImport("avifil32.dll")]
private static extern void AVIFileInit();
```

29.WHAT IS “SIZEOF” OPERATOR?

The **sizeof operator** returns the number of bytes occupied by a variable of a given type. The argument to the **sizeof operator** must be the name of an unmanaged type or a type parameter that is constrained to be an unmanaged type.

```
Console.WriteLine(sizeof(byte)); // output: 1
```

30. CAN WE USE “THIS” INSIDE A STATIC METHOD?

No. We can't use “this” in static method.

31. WHAT IS THE DIFFERENCE BETWEEN CTYPE AND DIRECTCAST?

- CType is used for conversion between type and the expression.
 - Directcast is used for converting the object type which requires run time type to be the same as specified type.
-

32. WHICH STRING METHOD IS USED FOR CONCATENATION OF TWO STRINGS?

“Concat” method of String class is used to concatenate two strings. For example,
string.Concat(firstStr, secStr)

33. WHAT IS PARSING? HOW TO PARSE A DATE TIME STRING?

Parsing converts a string into another data type.

For Example:

```
string text = "500";
int num = int.Parse(text);
```

500 is an integer. So, the Parse method converts the string 500 into its own base type, i.e int.

Follow the same method to convert a DateTime string.

```
string dateTime = "Jan 1, 2018";
DateTime parsedValue = DateTime.Parse(dateTime);
```

34. EXPLAIN PARTIAL CLASS?

Partial classes concept added in .Net Framework 2.0 and it allows us to split the business logic in multiple files with the same class name along with “partial” keyword.

35. EXPLAIN ANONYMOUS TYPE?

This is being added 3.0 version. This feature enables us to create an object at compile time. Below is the sample code for the same –

```
Var myTestCategory = new { CategoryId = 1, CategoryName = "Category1" };
```

36. EXPLAIN GET AND SET ACCESSOR PROPERTIES?

Get and Set are called Accessors. These are made use by Properties. The property provides a mechanism to read, write the value of a private field. For accessing that private field, these accessors are used.

Get Property is used to return the value of a property

Set Property accessor is used to set the value.

37. WHAT IS THE DIFFERENCE BETWEEN VAR AND DYNAMIC IN C#?

VAR TYPE - VAR are those variables which are declared without specifying the .NET type explicitly. In implicitly typed variable, the type of the variable is automatically deduced at compile time by the compiler from the value used to initialize the variable.

```
var a = 'f';
```

DYNAMIC TYPE - It is used to avoid the compile-time type checking. The compiler does not check the type of the dynamic type variable at compile time, instead of this, the compiler gets the type at the run time. The dynamic type variable is created using dynamic keyword.

```
dynamic val1 = "Interview";
```

38. WHAT IS COVARIANCE IN C#?

Covariance enables you to pass a derived type where a base type is expected.

```
Small sm1 = new Bigger();
```

.NET FRAMEWORK

39. [WHAT IS JIT?](#)

JIT stands for **Just In Time**. JIT is a compiler that converts Intermediate Language to a Native code.

The code is converted into Native language during execution. Native code is nothing but hardware specifications that can be read by the CPU. The native code can be stored so that it is accessible for subsequent calls.

40. [WHAT IS MSIL?](#)

MSIL stands for Microsoft Intermediate Language.

MSIL provides instructions for calling methods, initializing and storing values, operations such as memory handling, exception handling and so on. All .Net codes are first compiled to IL.

41. [WHAT IS MEANT BY MANAGED AND UNMANAGED CODE?](#)

The code that is managed by the CLR is called **Managed code**. This code runs inside the CLR. Hence, it is necessary to install the .Net framework in order to execute the managed code. CLR manages the memory through garbage collection and also uses the other features like CAS and CTS for efficient management of the code.

Unmanaged code is any code that does not depend on CLR for execution. It means it is developed by any other language independent of .Net framework. It uses its own runtime environment for compiling and execution.

Though it is not running inside the CLR, the unmanaged code will work properly if all the other parameters are correctly followed.

42. WHAT IS DIFFERENCE BETWEEN NAMESPACE AND ASSEMBLY?

Following are the differences between namespace and assembly:

- Assembly is physical grouping of logical units, Namespace, logically groups classes.
 - Namespace can span multiple assembly.
-

43. WHAT IS MANIFEST?

Assembly metadata is stored in Manifest. Manifest contains all the metadata needed to do the

following things (See Figure Manifest View for more details):

- Version of assembly.
 - Security identity.
 - Scope of the assembly.
 - Resolve references to resources and classes.
-

44. WHERE IS VERSION INFORMATION STORED OF AN ASSEMBLY?

Version information is stored in assembly inside the manifest.

45.IS VERSIONING APPLICABLE TO PRIVATE ASSEMBLIES?

Versioning concept is only applicable to global assembly cache (GAC) as private assembly lie in

their individual folders. This does not mean versioning is not needed , you can still version it to

have better version control on the project.

46.WHAT IS THE APPLICATION DOMAIN?

An Application Domain is a logical container for a set of assemblies in which an executable is hosted. As you have seen, a single process may contain multiple Application Domains, each of which is hosting a .NET executable.

47.EXPLAIN CAS (CODE ACCESS SECURITY).

.Net provides a security model that prevents unauthorized access to resources. CAS is a part of that security model. CAS is present in the CLR. It enables the users to set permissions at a granular level for the code.

48. WHAT ARE SYNCHRONOUS AND ASYNCHRONOUS OPERATIONS?

1. Synchronization is a way to create a thread-safe code where only one thread can access the resource at any given time. The asynchronous call waits for the method to complete before continuing with the program flow.
2. Synchronous programming badly affects the UI operations when the user tries to perform time-consuming operations since only one thread will be used. In Asynchronous operation, the method call will immediately return so

that the program can perform other operations while the called method completes its work in certain situations.

49. WHAT IS MULTI-TASKING?

It is a feature of modern operating systems with which we can run multiple programs at same time example Word, Excel etc.

50. NAME SOME PROPERTIES OF THREAD CLASS.

Few Properties of thread class are:

- **IsAlive** – contains value True when a thread is Active.
 - **Name** – Can return the name of the thread. Also, can set a name for the thread.
 - **Priority** – returns the prioritized value of the task set by the operating system.
 - **IsBackground** – gets or sets a value which indicates whether a thread should be a background process or foreground.
 - **ThreadState** – describes the thread state.
-

51. WHAT ARE THE DIFFERENT STATES OF A THREAD?

Different states of a thread are:

- **Unstarted** – Thread is created.
- **Running** – Thread starts execution.
- **WaitSleepJoin** – Thread calls sleep, calls wait on another object and calls join on another thread.
- **Suspended** – Thread has been suspended.

- **Aborted** – Thread is dead but not changed to state stopped.
 - **Stopped** – Thread has stopped.
-

52. CAN WE HAVE MULTIPLE THREADS IN ONE APP DOMAIN?

One or more threads run in an AppDomain. An AppDomain is a runtime representation of a

logical process within a physical process. Each AppDomain is started with a single thread, but

can create additional threads from any of its threads.

53. WHICH NAMESPACE HAS THREADING?

‘Systems.Threading’ has all the classes related to implement threading.

54. EXPLAIN LOCK, MONITORS, AND MUTEX OBJECT IN THREADING.

Lock keyword ensures that only one thread can enter a particular section of the code at any given time. In the above Example, lock(ObjA) means the lock is placed on ObjA until this process releases it, no other thread can access ObjA.

Mutex is also like a lock but it can work across multiple processes at a time. WaitOne() is used to lock and ReleaseMutex() is used to release the lock. But Mutex is slower than lock as it takes time to acquire and release it.

Monitor.Enter and Monitor.Exit implements lock internally. a lock is a shortcut for Monitors. lock(objA) internally calls.

Monitor.Enter(ObjA);

try

```
{  
}  
Finally {Monitor.Exit(ObjA)};
```

55. WHAT IS THREAD.SLEEP () IN THREADING?

Thread's execution can be paused by calling the Thread.Sleep method. This method takes an integer value that determines how long the thread should sleep. Example Thread.CurrentThread.Sleep(2000).

56. WHAT IS SUSPEND AND RESUME IN THREADING?

Suspend() method is called to suspend the thread. Resume() method is called to resume the suspended thread. Start() method is used to send a thread into runnable State.

57. WHAT THE WAY TO STOP A LONG RUNNING THREAD?

Thread.Abort() stops the thread execution at that moment itself.

58. WHY WE NEED MULTI-THREADING IN OUR PROJECT?

Multi-threading is running the multiple threads simultaneously. Some main advantages are:

- You can do multiple tasks simultaneously. For e.g. saving the details of user to a file while at the same time retrieving something from a web service.
 - Threads are much lightweight than process. They don't get their own resources. They used the resources allocated to a process.
 - Context-switch between threads takes less time than process.
-

59. HOW TO START A THREAD IN C#?

We have to use the Thread class provided by System.Threading namespace. In the constructor of the class, we have to pass the method name which we want to run in separate thread. After than we have to call the start method of Thread class. Below is the example.

60. WHAT IS RACE CONDITION?

A race condition happens when two or more threads want to update shared data at the same time.

61. WHAT IS LIVELOCK?

A livelock is very similar to deadlock except involved threads states are continually changing their state but still they cannot complete their work.

A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but

they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.

62. WHAT IS IMMUTABLE OBJECT?

An immutable object is an object whose state cannot be changed after creation.

Immutable objects are useful in concurrent programming as they are thread-safe.

"String" objects are examples of immutable objects because we cannot change the value of string after it is created.

63. HOW MANAGED CODE IS EXECUTED?

Managed code is a code whose execution is managed by Common Language Runtime. It gets the managed code and compiles it into machine code. After that, the code is executed. The runtime here i.e. CLR provides automatic memory management, type safety, etc.

64. WHAT IS THE DIFFERENCE BETWEEN SYSTEM EXCEPTIONS AND APPLICATION EXCEPTIONS?

In general System exceptions occur whenever some non-recoverable or fatal error is encountered, like a database crash, bound errors etc.

While in case of Application level exceptions some error which is recoverable is encountered, for instance, the wrong type of input data, arithmetic exceptions etc.

65. WHAT IS XSD?

XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document. It can be used by programmers to verify each piece of item content in a document.

ASP.NET MVC

66. IN WHICH ASSEMBLY IS THE MVC FRAMEWORK DEFINED?

System.Web.Mvc

67. IS IT POSSIBLE TO SHARE A VIEW ACROSS MULTIPLE CONTROLLERS?

Yes, put the view into the shared folder. This will automatically make the view available across multiple controllers.

68. WHAT IS ROUTE IN MVC? WHAT IS DEFAULT ROUTE IN MVC?

A route is a URL pattern that is mapped to a handler. The handler can be a physical file, such as a .aspx file in a Web Forms application. A handler can also be a class that processes the request, such as a controller in an MVC application. To define a route, you create an instance of the [Route](#) class by specifying the URL pattern, the handler, and optionally a name for the route.

You add the route to the application by adding the Route object to the static Routes property of the RouteTable class. The Routes property is a RouteCollection object that stores all the routes for the application.

You typically do not have to write code to add routes in an MVC application. Visual Studio project templates for MVC include preconfigured URL routes. These are defined in the MVC Application class, which is defined in the Global.asax file.

Default Route

The default ASP.NET MVC project templates add a generic route that uses the following URL convention to break the URL for a given request into three named segments.

URL: "{controller}/{action}/{id}"

This route pattern is registered via a call to the `MapRoute()` extension method of `RouteCollection`.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(      Ignore routes that end with axd extension
                        name: "Default",      Route Name
                        url: "{controller}/{action}/{id}",      URL Pattern
                        defaults: new { controller = "Home", action = "Index",
                            id = UrlParameter.Optional }      Default Values
    );
}
```

69. WHERE ARE THE ROUTING RULES DEFINED IN AN ASP.NET MVC APPLICATION?

In `Application_Start` event in `Global.asax`

70. WHAT IS VALIDATION SUMMARY IN MVC?

The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.

The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages.

71. DIFFERENCES BETWEEN RAZOR AND ASPX VIEW ENGINE IN MVC?

Razor View Engine VS ASPX View Engine

The Razor View Engine is a bit slower than the ASPX View Engine.

Conclusion

Razor provides a new view engine with a streamlined code for focused templating. Razor's syntax is very compact and improves the readability of the markup and code. By default, MVC supports ASPX (web forms) and Razor View Engine. MVC also supports third-party view engines like Spark, Nhaml, NDjango, SharpDOM and so on. ASP.NET MVC is open source.

72. WHAT ARE THE MAIN RAZOR SYNTAX RULES?

- Razor code blocks are enclosed in @{ ... }
- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword

- Strings are enclosed with quotation marks
 - C# code is case sensitive
 - C# files have the extension .cshtml
-

73. HOW DO YOU IMPLEMENT FORMS AUTHENTICATION IN MVC?

Authentication is giving access to the user for a specific service by verifying his/her identity using his/her credentials like username and password or email and password. It assures that the correct user is authenticated or logged in for a specific service and the right service has been provided to the specific user based on their role that is nothing but authorization.

ASP.NET forms authentication occurs after IIS authentication is completed. You can configure forms authentication by using forms element with in web.config file of your application. The default attribute values for forms authentication are shown below,

```
<system.web>
  <authenticationmode="Forms">
    <formsloginUrl="Login.aspx" protection="All" timeout="30" name=".ASPXAUTH" path="/" requireSSL="false" slidingExpiration="true" defaultUrl="default.aspx" cookieless="UseDeviceProfile" enableCrossAppRedirects="false"/>
  </authentication>
</system.web>
```

The FormsAuthentication class creates the authentication cookie automatically when SetAuthCookie() or RedirectToLoginPage() methods are called. The value of authentication cookie contains a string representation of the encrypted and signed FormsAuthenticationTicket object.

74. HOW WE CAN REGISTER THE AREA IN ASP.NET MVC?

When we have created an area make sure this will be registered in “Application_Start” event in Global.asax. Below is the code snippet where area registration is done :

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
}
```

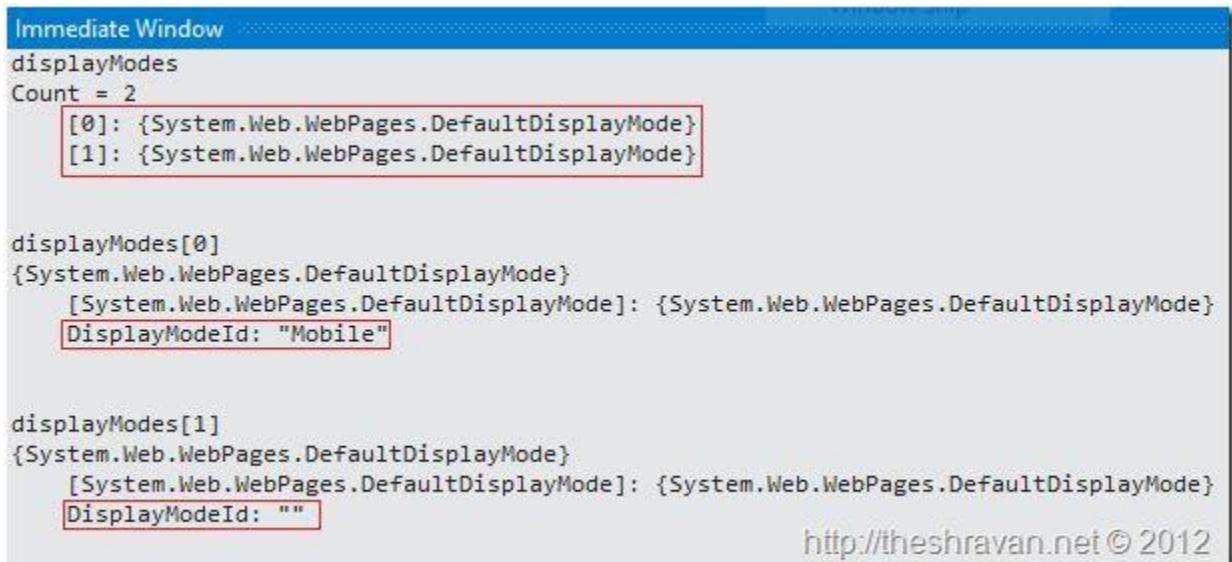
75. EXPLAIN THE NEED OF DISPLAY MODE IN MVC?

DisplayModes give you another level of flexibility on top of the default capabilities we saw in the last section. DisplayModes can also be used along with the previous feature so we will simply build off of the site we just created.

Using display modes involves in 2 steps

1. We should register Display Mode with a suffix for particular browser using “DefaultDisplayMode” class in Application_Start() method in the Global.asax file.

2. View name for particular browser should be appended with suffix mentioned in first step.



The screenshot shows the Immediate Window in Visual Studio. It displays the following code and its execution results:

```
displayModes
Count = 2
[0]: {System.Web.WebPages.DefaultDisplayMode}
[1]: {System.Web.WebPages.DefaultDisplayMode}

displayModes[0]
{System.Web.WebPages.DefaultDisplayMode}
    [System.Web.WebPages.DefaultDisplayMode]: {System.Web.WebPages.DefaultDisplayMode}
    DisplayModeId: "Mobile"

displayModes[1]
{System.Web.WebPages.DefaultDisplayMode}
    [System.Web.WebPages.DefaultDisplayMode]: {System.Web.WebPages.DefaultDisplayMode}
    DisplayModeId: ""
```

http://theshravan.net © 2012

1. Desktop browsers (without any suffix. e.g.: Index.cshtml, _Layout.cshtml).
2. Mobile browsers (with a suffix “Mobile”. e.g.:
Index.Mobile.cshtml, Layout.Mobile.cshtml)

If you want design different pages for different mobile device browsers (any different browsers) and render them depending on the browser requesting. To handle these requests you can register custom display modes. We can do that using `DisplayModeProvider.Instance.Modes.Insert(int index, IDisplayMode item)` method.

76. WHAT IS ROUTE CONSTRAINTS IN MVC?

Routing is a great feature of MVC, it provides a REST based URL that is very easy to remember and improves page ranking in search engines.

This article is not an introduction to Routing in MVC, but we will learn a few features of routing and by implementing them we can develop a very flexible and user-friendly application. So, let's start without wasting valuable time.

Add constraint to URL

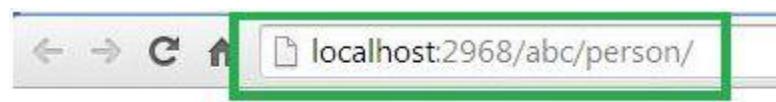
This is very necessary for when we want to add a specific constraint to our URL. Say, for example we want a [URL](#).

So, we want to set some constraint string after our host name. Fine, let's see how to implement it.

It's very simple to implement, just open the RouteConfig.cs file and you will find the routing definition in that. And modify the routing entry as in the following. We will see that we have added "abc" before.

```
17     routes.MapRoute(18         name: "Default",19         url: "abc/{controller}/{action}/{id}",20         defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }21     );22 }23 }
```

Controller name, now when we browse we need to specify the string in the URL, as in the following:



I am simple view.

77. WHAT ARE THE FOLDERS IN MVC APPLICATION SOLUTIONS?

Understanding the folders

When you create a project a folder structure gets created by default under the name of your project which can be seen in solution explorer. Below i will give you a brief explanation of what these folders are for.

Model

This folder contains classes that is used to provide data. These classes can contain data that is retrieved from the database or data inserted in the form by the user to update the database.

Controllers

These are the classes which will perform the action invoked by the user. These classes contains methods known as "Actions" which responds to the user action accordingly.

Views

These are simple pages which uses the model class data to populate the HTML controls and renders it to the client browser.

App_Start

Contains Classes such as FilterConfig, RoutesConfig, WebApiConfig. As of now we need to understand the RouteConfig class. This class contains the default format of the url that should be supplied in the browser to navigate to a specified page.

78. WHAT ARE THE METHODS OF HANDLING AN ERROR IN MVC?

Exception handling may be required in any application, whether it is a web application or a Windows Forms application.

ASP.Net MVC has an attribute called "HandleError" that provides built-in exception filters. The HandleError attribute in ASP.NET MVC can be applied over the action method as well as Controller or at the global level. The HandleError attribute is the default implementation of IExceptionFilter. When we create a MVC application, the HandleError attribute is added within the Global.asax.cs file and registered in the Application_Start event.

```
1. public static void RegisterGlobalFilters(GlobalFilterCollection filters)  
2. {  
3.     filters.Add(new HandleErrorAttribute());  
4. }  
5. protected void Application_Start()  
6. {  
7.     AreaRegistration.RegisterAllAreas();  
8.     RegisterGlobalFilters(GlobalFilters.Filters);  
9.     RegisterRoutes(RouteTable.Routes);  
10.}
```

Important properties of HandleError attribute

The HandleError Error attribute has a couple for properties that are very useful in handling the exception.

ExceptionType

Type of exception to be catch. If this property is not specified then the HandleError filter handles all exceptions.

View

Name of the view page for displaying the exception information.

Master

Master View for displaying the exception.

Order

Order in which the action filters are executed. The Order property has an integer value and it specifies the priority from 1 to any positive integer value. 1 means highest priority and the greater the value of the integer is, the lower is the priority of the filter.

AllowMultiple

It indicates whether more than one instance of the error filter attribute can be specified.

Example

```
[HandleError(View = "Error")]

public class HomeController: Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Welcome to ASP.NET MVC!";
        int u = Convert.ToInt32(""); // Error line
        return View();
    }
}
```

HandleError Attribute at Action Method Level,

```
[HandleError(View = "Error")]

public ActionResult Index()
{
    ViewBag.Message = "Welcome to ASP.NET MVC!";
    int u = Convert.ToInt32(""); // Error line
    return View();
}
```

79.WHAT IS VIEWSTART?

Razor View Engine introduced a new layout named _ViewStart which is applied on all view automatically. Razor View Engine firstly executes the _ViewStart and then start rendering the other view and merges them.

Example of Viewstart

```
@ {  
    Layout = "~/Views/Shared/_v1.cshtml";  
}  
<!DOCTYPE html>  
<html>  
<head>  
<meta name = "viewport"  
content = "width=device-width" />  
<title>ViewStart</title></head><body>  
</body></html>
```

80.HOW CAN WE DONE CUSTOM ERROR PAGE IN MVC?

The HandleErrorAttribute allows you to use a custom page for this error. First you need to update your web.config file to allow your application to handle custom errors.

1. <system.web>
2. <customErrors mode="On">
3. </system.web>

Then, your action method needs to be marked with the `attribute`.

1. `[HandleError]`
2. `public class HomeController: Controller`
3. `{`
4. `[HandleError]`
5. `public ActionResult ThrowException()`
6. `{`
7. `throw new ApplicationException();`
8. `}`
9. `}`

By calling the `ThrowException` action, this would then redirect the user to the default error page. In our case though, we want to use a custom error page and redirect the user there instead. So, let's create our new custom view page.

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CustomErrorView</title>
</head>
<body>
    <h2>
        Error</h2>
    <p>
        Controller:
        <%= ((HandleErrorInfo)ViewData.Model).ControllerName %>
    </p>
    <p>
        Action:
        <%= ((HandleErrorInfo)ViewData.Model).ActionName %>
    </p>
    <p>
        Message:
        <%= ((HandleErrorInfo)ViewData.Model).Exception.Message %>
    </p>
    <p>
        Stack Trace:
        <%= ((HandleErrorInfo)ViewData.Model).Exception.StackTrace %>
    </p>
</body>
</html>
```

Next, we simply need to update the HandleErrorAttribute on the action method.

```
1. [HandleError]
2. public class HomeController: Controller
3. {
4.     [HandleError(View = "CustomErrorView")]
5.     public ActionResult ThrowException()
6.     {
7.         throw new ApplicationException();
8.     }
9. }
```

81. WHAT IS SERVER SIDE VALIDATION IN MVC?

The ASP.NET MVC Framework validates any data passed to the controller action that is executing. It populates a ModelState object with any validation failures that it finds and passes that object to the controller. Then the controller actions can query the ModelState to discover whether the request is valid and react accordingly.

I will use two approaches in this article to validate a model data. One is to manually add an error to the ModelState object and another uses the Data Annotation API to validate the model data.

Approach 1 - Manually Add Error to ModelState object

I create a User class under the Models folder. The User class has two properties "Name" and "Email". The "Name" field has required field validations while the "Email" field has Email validation. So let's see the procedure to implement the validation. Create the User Model as in the following,

```
1. namespace ServerValidation.Models
2. {
3.   public class User
4.   {
5.     public string Name
6.     {
7.       get;
8.       set;
9.     }
10.    public string Email
11.    {
12.      get;
13.      set;
14.    }
15.  }
16.}
```

After that I create a controller action in User Controller (UserController.cs under Controllers folder). That action method has logic for the required validation for Name and Email validation on the Email field. I add an error message on ModelState with a key and that message will be shown on the view whenever the data is not to be validated in the model.

```
1. using System.Text.RegularExpressions;
2. using System.Web.Mvc;
3. namespace ServerValidation.Controllers
4. {
5.   public class UserController: Controller
6.   {
7.     public ActionResult Index()
8.     {
9.       return View();
10.    }
11. }
```

```

11.    [HttpPost]
12.    public ActionResult Index(ServerValidation.Models.User model)
13.    {
14.
15.        if (string.IsNullOrEmpty(model.Name))
16.        {
17.            ModelState.AddModelError("Name", "Name is required");
18.        }
19.        if (!string.IsNullOrEmpty(model.Email))
20.        {
21.            string emailRegex = @ "^(a-zA-Z0-9_\-\.)+@[a-zA-Z0-9]{1,3}\." +
22.                @ ".[0-9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-
23.                ]+)\." +
24.                @ ".)+)([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$";
25.            Regex re = new Regex(emailRegex);
26.            if (!re.IsMatch(model.Email))
27.            {
28.                ModelState.AddModelError("Email", "Email is not val
29.                id");
30.            }
31.        } else {
32.            ModelState.AddModelError("Email", "Email is required")
33.        ;
34.        }
35.        if (ModelState.IsValid)
36.        {
37.            ViewBag.Name = model.Name;
38.            ViewBag.Email = model.Email;
39.        }
40.    }

```

Thereafter I create a view (Index.cshtml) for the user input under the User folder.

```

1. @model ServerValidation.Models.User
2. @{
3.     ViewBag.Title = "Index";
4. }
5. @using(Html.BeginForm())
6. {
7.     if (@ViewData.ModelState.IsValid)
8.     {
9.         if (@ViewBag.Name != null)
10.        { <b>
11.            Name: @ViewBag.Name <br />
12.            Email: @ViewBag.Email </b>
13.        }
14.    } <fieldset>
15.    <legend>User </legend> <div class = "editor-label" >
16.        @Html.LabelFor(model => model.Name) </div> <div class = "edi
tor-field" >
17.        @Html.EditorFor(model => model.Name)
18.    @if(!ViewData.ModelState.IsValid)
19.    {
20.        <span class = "field-validation-
error" > @ViewData.ModelState["Name"].Errors[0].ErrorMessage </spa
n>
21.
22.    }
23.    </div> <div class = "editor-label" >
24.
25.        @Html.LabelFor(model => model.Email) </div> <div class = "edi
tor-field" >
26.        @Html.EditorFor(model => model.Email)
27.    @if(!ViewData.ModelState.IsValid)
28.    {
29.        <span class = "field-validation-
error" > @ViewData.ModelState["Email"].Errors[0].ErrorMessage </sp
an>
30.    }

```

```
31. </div><p>
32.   <input type = "submit"
33.   value = "Create" />
34. </p></fieldset>
35.}
```

82. WHAT IS THE USE OF REMOTE VALIDATION IN MVC?

Remote validation is the process where we validate specific data posting data to a server without posting the entire form data to the server. Let's see an actual scenario, in one of my projects I had a requirement to validate an email address, whether it already exists in the database. Remote validation was useful for that; without posting all the data we can validate only the email address supplied by the user.

Practical Explanation

Let's create a MVC project and name it accordingly, for me its "TestingRemoteValidation". Once the project is created let's create a model named UserModel that will look like:

```
1. public class UserModel
2. {
3.   [Required]
4.   public string UserName
5.   {
6.     get;
7.     set;
8.   }
9.   [Remote("CheckExistingEmail", "Home", ErrorMessage = "Email
already exists!")]

```

```

10. public string UserEmailAddress
11. {
12.     get;
13.     set;
14. }
15.

```

Let's get some understanding of the remote attribute used, so the very first parameter "CheckExistingEmail" is the the name of the action. The second parameter "Home" is referred to as controller so to validate the input for the UserEmailAddress the "CheckExistingEmail" action of the "Home" controller is called and the third parameter is the error message. Let's implement the "CheckExistingEmail" action result in our home controller.

```

1. public ActionResult CheckExistingEmail(string UserEmailAddress)
2. {
3.     bool ifEmailExist = false;
4.     try
5.     {
6.         ifEmailExist = UserEmailAddress.Equals("mukeshknayak@gmail.com") ? true : false;
7.         return Json(!ifEmailExist, JsonRequestBehavior.AllowGet);
8.     } catch (Exception ex)
9.     {
10.        return Json(false, JsonRequestBehavior.AllowGet);
11.    }
12.

```

83. EXPLAIN RENDERSECTION IN MVC?

RenderSection() is a method of the WebPageBase class. Scott wrote at one point, The first parameter to the "RenderSection()" helper method specifies the name of the section we want to render at that location in the layout template. The second

parameter is optional, and allows us to define whether the section we are rendering is required or not. If a section is "required", then Razor will throw an error at runtime if that section is not implemented within a view template that is based on the layout file (that can make it easier to track down content errors). It returns the HTML content to render.

```
1. <div id="body">
2.   @RenderSection("featured", required: false)
3.   <section class="content-wrapper main-content clear-
   fix">
4.     @RenderBody()
5.   </section>
6. </div>
```

84. WHAT IS THE SIGNIFICANCE OF NONACTIONATTRIBUTE?

In general, all public methods of a controller class are treated as action methods. If you want prevent this default behavior, just decorate the public method with NonActionAttribute.

85. HOW ROUTE TABLE IS CREATED IN ASP.NET MVC?

When an MVC application first starts, the Application_Start() method is called. This method, in turn, calls the RegisterRoutes() method. The RegisterRoutes() method creates the route table.

86. ASP.NET MVC APPLICATION, MAKES USE OF SETTINGS AT 2 PLACES FOR ROUTING TO WORK CORRECTLY. WHAT ARE THESE 2 PLACES?

1. Web.Config File : ASP.NET routing has to be enabled here.
 2. Global.asax File : The Route table is created in the application Start event handler, of the Global.asax file.
-

87. WHAT IS THE USE OF THE FOLLOWING DEFAULT ROUTE?

{resource}.axd/{*pathInfo}

This route definition, prevent requests for the Web resource files such as WebResource.axd or ScriptResource.axd from being passed to a controller.

88. WHAT IS THE DIFFERENCE BETWEEN ADDING ROUTES, TO A WEBFORMS APPLICATION AND TO AN MVC APPLICATION?

To add routes to a webforms application, we use MapPageRoute() method of the RouteCollection class, where as to add routes to an MVC application we use MapRoute() method.

89. IF I HAVE MULTIPLE FILTERS IMPLEMENTED, WHAT IS THE ORDER IN WHICH THESE FILTERS GET EXECUTED?

1. Authorization filters
 2. Action filters
 3. Response filters
 4. Exception filters
-

90. WHICH FILTER EXECUTES FIRST IN AN ASP.NET MVC APPLICATION?

Authorization filter

91. WHAT ARE THE LEVELS AT WHICH FILTERS CAN BE APPLIED IN AN ASP.NET MVC APPLICATION?

1. Action Method
 2. Controller
 3. Application
-

92. IS IT POSSIBLE TO CREATE A CUSTOM FILTER?

Yes

93. WHAT FILTERS ARE EXECUTED IN THE END?

Exception Filters

94. IS IT POSSIBLE TO CANCEL FILTER EXECUTION?

Yes

95. WHAT TYPE OF FILTER DOES OUTPUTCACHEATTRIBUTE CLASS REPRESENTS?

Result Filter

96. WHAT ARE THE 2 POPULAR ASP.NET MVC VIEW ENGINES?

-
1. Razor
 2. .aspx
-

97. WHAT SYMBOL WOULD YOU USE TO DENOTE, THE START OF A CODE BLOCK IN RAZOR VIEWS?

@

98. WHAT SYMBOL WOULD YOU USE TO DENOTE, THE START OF A CODE BLOCK IN ASPX VIEWS?

<%= %>

In razor syntax, what is the escape sequence character for @ symbol?

The escape sequence character for @ symbol, is another @ symbol

99. WHAT ARE SECTIONS?

Layout pages, can define sections, which can then be overriden by specific views making use of the layout. Defining and overriding sections is optional.

100. WHAT ARE THE FILE EXTENSIONS FOR RAZOR VIEWS?

1. .cshtml – If the programming lanugaue is C#
 2. .vbhtml – If the programming lanugaue is VB
-

101. HOW DO YOU SPECIFY COMMENTS USING RAZOR SYNTAX?

Razor syntax makes use of @* to indicate the begining of a comment and *@ to indicate the end.

102. IS IT POSSIBLE TO COMBINE ASP.NET WEBFORMS AND ASP.MVC AND DEVELOP A SINGLE WEB APPLICATION?

Yes, it is possible to combine ASP.NET webforms and ASP.MVC and develop a single web application.

103. WHAT IS THE USE OF VIEWMODEL IN MVC?

ViewModel is a plain class with properties, which is used to bind it to strongly typed view. ViewModel can have the validation rules defined for its properties using data annotations.

104. EXPLAIN BUNDLE.CONFIG IN MVC4?

“BundleConfig.cs” in MVC4 is used to register the bundles by the bundling and minification system. Many bundles are added by default including jQuery libraries like – jquery.validate, Modernizr, and default CSS references.

105. HOW WE CAN HANDLE THE EXCEPTION AT CONTROLLER LEVEL IN ASP.NET MVC?

Exception Handling is made simple in ASP.Net MVC and it can be done by just overriding “OnException” and set the result property of the filtercontext object (as shown below) to the view detail, which is to be returned in case of exception.

```
protected overrides void OnException(ExceptionContext filterContext)
{
}
```

106. DOES TEMPDATA HOLD THE DATA FOR OTHER REQUEST IN ASP.NET MVC?

If TempData is assigned in the current request then it will be available for the current request and the subsequent request and it depends whether data in TempData read or not. If data in TempData is read then it would not be available for the subsequent requests.

107. EXPLAIN KEEP METHOD IN TEMPDATA IN ASP.NET MVC?

As explained above in case data in TempData has been read in current request only then “Keep” method has been used to make it available for the subsequent request.

```
@TempData[“TestData”];
TempData.Keep(“TestData”);
```

108. EXPLAIN PEEK METHOD IN TEMPDATA IN ASP.NET MVC?

Similar to Keep method we have one more method called “Peek” which is used for the same purpose. This method used to read data in TempData and it maintains the data for subsequent request.

```
string A4str = TempData.Peek(“TT”).ToString();
```

109. WHAT ARE CHILD ACTIONS IN ASP.NET MVC?

To create reusable widgets child actions are used and this will be embedded into the parent views. In ASP.Net MVC Partial views are used to have reusability in the application. Child action mainly returns the partial views.

110. EXPLAIN THE TOOLS USED FOR UNIT TESTING IN ASP.NET MVC?

Below are the tools used for unit testing :

NUnit
xUnit.NET
Ninject 2
Moq

111. CAN I USE RAZOR CODE IN JAVASCRIPT IN ASP.NET MVC?

Yes. We can use the razor code in javascript in cshtml by using <text> element.

```
< script type="text/javascript">
@foreach (var item in Model) {
< text >
//javascript goes here which uses the server values
< text >
}
< script>
```

112. HOW CAN I RETURN STRING RESULT FROM ACTION IN ASP.NET MVC?

Below is the code snippet to return string from action method :

```
public ActionResult TestAction() {  
    return Content("Hello Test !!");  
}
```

113. HOW TO RETURN THE JSON FROM ACTION METHOD IN ASP.NET MVC?

Below is the code snippet to return string from action method :

```
public ActionResult TestAction() {  
    return JSON(new { prop1 = "Test1", prop2 = "Test2" });  
}
```

114. GIVE AN EXAMPLE FOR AUTHORIZATION FILTERS IN AN ASP.NET MVC APPLICATION?

1. `RequireHttpsAttribute`
 2. `AuthorizeAttribute`
-

115. WHAT IS DATA ANNOTATION VALIDATOR ATTRIBUTES IN MVC?

Data Annotations are nothing but certain validations that we put in our models to validate the input from the user. ASP.NET **MVC** provides a unique feature in which we can validate the models using the ***Data Annotation attribute***. Import the following namespace to use ***data annotations*** in the application.

Example

[Required(ErrorMessage = "Please enter name"), MaxLength(30)]

[Display(Name = "Student Name")]

```
public string Name { get; set; }
```

116. WHAT ARE THE EXCEPTION FILTERS IN MVC?

Exception filter in MVC provides an ability to handle the exceptions for all the controller methods at a single location. This is by creating a class, which inherits from the FilterAttribute and IExceptionFilter interface.

117. WHICH APPROACH PROVIDES BETTER SUPPORT FOR TEST DRIVEN DEVELOPMENT - ASP.NET MVC OR ASP.NET WEBFORMS?

ASP.NET MVC

ASP.NET WEBFORMS

118. WHAT'S THE USE OF RESPONSE.OUTPUT.WRITE()?

We can write formatted output using Response.Output.Write().

119. IN WHICH EVENT OF PAGE CYCLE IS THE VIEWSTATE AVAILABLE?

After the Init() and before the Page_Load().

120. FROM WHICH BASE CLASS ALL WEB FORMS ARE INHERITED?

Page class.

121. WHICH VALIDATOR CONTROL YOU USE IF YOU NEED TO MAKE SURE THE VALUES IN TWO DIFFERENT CONTROLS MATCHED?

Compare Validator control.

122. WHAT IS VIEWSTATE?

ViewState is used to retain the state of server-side objects between page post backs.

123. HOW LONG THE ITEMS IN VIEWSTATE EXISTS?

They exist for the life of the current page.

124. WHAT ARE THE DIFFERENT VALIDATORS IN ASP.NET?

1. Required field Validator
 2. Range Validator
 3. Compare Validator
 4. Custom Validator
 5. Regular expression Validator
 6. Summary Validator
-

125. HOW YOU CAN ADD AN EVENT HANDLER?

Using the Attributes property of server side control.

e.g.

```
btnSubmit.Attributes.Add("onMouseOver", "JavascriptCode();")
```

126. WHICH TYPE OF CACHING WILL BE USED IF WE WANT TO CACHE THE PORTION OF A PAGE INSTEAD OF WHOLE PAGE?

Fragment Caching: It caches the portion of the page generated by the request. For that, we can create user controls with the below code:

```
<%@ OutputCache Duration="120" VaryByParam="CategoryID;SelectedID"%>
```

127. CAN WE HAVE A WEB APPLICATION RUNNING WITHOUT WEB.CONFIG FILE?

Yes

128. CAN WE ADD CODE FILES OF DIFFERENT LANGUAGES IN APP_CODE FOLDER?

No. The code files must be in same language to be kept in App_code folder.

129. WHAT IS PROTECTED CONFIGURATION?

It is a feature used to secure connection string information.

130. WRITE CODE TO SEND E-MAIL FROM AN ASP.NET APPLICATION?

```
MailMessage mailMess = new MailMessage();  
mailMess.From = "abc@gmail.com";  
mailMess.To = "xyz@gmail.com";  
mailMess.Subject = "Test email";  
mailMess.Body = "Hi This is a test mail.;"
```

```
SmtpMail.SmtpServer = "localhost";
```

```
SmtpMail.Send (mailMess);
```

MailMessage and SmtpMail are classes defined System.Web.Mail namespace.

131. HOW CAN WE PREVENT BROWSER FROM CACHING AN ASPX PAGE?

We can SetNoStore on HttpCachePolicy object exposed by the Response object's Cache property:

```
Response.Cache.SetNoStore ();
```

```
Response.Write (DateTime.Now.ToString ("o"));
```

132. WHAT IS THE GOOD PRACTICE TO IMPLEMENT VALIDATIONS IN ASPX PAGE?

Client-side validation is the best way to validate data of a web page. It reduces the network traffic and saves server resources.

133. WHAT ARE THE EVENT HANDLERS THAT WE CAN HAVE IN GLOBAL.ASAX FILE?

Application Events: Application_Start , Application_End, Application_AcquireRequestState, Application_AuthenticateRequest, Application_AuthorizeRequest, Application_BeginRequest, Application_Disposed, Application_EndRequest, Application_Error, Application_PostRequestHandlerExecute, Application_PreRequestHandlerExecute, Application_PreSendRequestContent,

Application_PresendRequestHeaders, Application_ReleaseRequestState,
Application_ResolveRequestCache, Application_UpdateRequestCache

Session Events: Session_Start, Session_End

134. WHICH PROTOCOL IS USED TO CALL A WEB SERVICE?

HTTP Protocol

135. EXPLAIN ROLE BASED SECURITY?

Role Based Security used to implement security based on roles assigned to user groups in the organization.

Then we can allow or deny users based on their role in the organization. Windows defines several built-in groups, including Administrators, Users, and Guests.

```
<AUTHORIZATION><authorization>

<allow roles="Domain_Name\Administrators" /> <!-- Allow Administrators in
domain. -- >

<deny users="*" /> <!-- Deny anyone else. -- >

</authorization>
```

136. HOW CAN WE APPLY THEMES TO AN ASP.NET APPLICATION?

We can specify the theme in web.config file. Below is the code example to apply theme:

```
<configuration>
```

```
<system.web>  
<pages theme="Windows7" />  
</system.web>  
</configuration>
```

137. WHAT IS REDIRECTPERMANENT IN ASP.NET?

RedirectPermanent Performs a permanent redirection from the requested URL to the specified URL. Once the redirection is done, it also returns 301 Moved Permanently responses.

138. EXPLAIN THE WORKING OF PASSPORT AUTHENTICATION.

First of all it checks passport authentication cookie. If the cookie is not available then the application redirects the user to Passport Sign on page. Passport service authenticates the user details on sign on page and if valid then stores the authenticated cookie on client machine and then redirect the user to requested page

139. WHAT ARE THE ADVANTAGES OF PASSPORT AUTHENTICATION?

All the websites can be accessed using single login credentials. So no need to remember login credentials for each web site.

Users can maintain his/ her information in a single location.

140. WHAT ARE THE ASP.NET SECURITY CONTROLS?

- <asp:Login>: Provides a standard login capability that allows the users to enter their credentials
 - <asp:LoginName>: Allows you to display the name of the logged-in user
 - <asp:LoginStatus>: Displays whether the user is authenticated or not
 - <asp:LoginView>: Provides various login views depending on the selected template
 - <asp:PasswordRecovery>: email the users their lost password
-

141. HOW DO YOU REGISTER JAVASCRIPT FOR WEBCONTROLS ?

We can register javascript for controls using <CONTROL - name>Attribtues.Add(scriptname,scripttext) method.

142. DIFFERENTIATE STRONG TYPING AND WEAK TYPING

In strong typing, the data types of variable are checked at compile time. On the other hand, in case of weak typing the variable data types are checked at runtime. In case of strong typing, there is no chance of compilation error. Scripts use weak typing and hence issues arises at runtime.

143. LIST ALL TEMPLATES OF THE REPEATER CONTROL.

- ItemTemplate
- AlternatingItemTemplate
- SeparatorTemplate
- HeaderTemplate
- FooterTemplate

144. LIST THE MAJOR BUILT-IN OBJECTS IN ASP.NET?

- Application
 - Request
 - Response
 - Server
 - Session
 - Context
 - Trace
-

145. WHAT IS THE APPSETTINGS SECTION IN THE WEB.CONFIG FILE?

The appSettings block in web config file sets the user-defined values for the whole application.

For example, in the following code snippet, the specified ConnectionString section is used throughout the project for database connection:

```
<em><configuration>

<appSettings>

<add key="ConnectionString" value="server=local; pwd=password;
database=default" />

</appSettings></em>
```

146. WHICH NAMESPACES ARE NECESSARY TO CREATE A LOCALIZED APPLICATION?

System.Globalization

147. WHAT ARE THE DIFFERENT TYPES OF COOKIES IN ASP.NET?

Session Cookie - Resides on the client machine for a single session until the user does not log out.

Persistent Cookie - Resides on a user's machine for a period specified for its expiry, such as 10 days, one month, and never.

148. WHAT IS THE FILE EXTENSION OF WEB SERVICE?

Web services have file extension .asmx.

149. IS IT POSSIBLE TO CREATE WEB APPLICATION WITH BOTH WEBFORMS AND MVC?

Yes. We have to include below mvc assembly references in the web forms application to create hybrid application.

System.Web.Mvc

System.Web.Razor

System.ComponentModel.DataAnnotations

150. CAN WE HAVE MULTIPLE WEB CONFIG FILES FOR AN ASP.NET APPLICATION?

Yes, one can create different folders and create a config for the specific folder.

151. WHAT IS THE DIFFERENCE BETWEEN WEB CONFIG AND MACHINE CONFIG?

Web config file is specific to a web application where as machine config is specific to a machine or server. There can be multiple web config files into an application where as we can have only one machine config file on a server.

Web config fields will override machine configurations.

152. WHAT IS CROSS PAGE POSTING?

When we click submit button on a web page, the page post the data to the same page. The technique in which we post the data to different pages is called Cross Page posting. This can be achieved by setting POSTBACKURL property of the button that causes the postback. Findcontrol method of PreviousPage can be used to get the posted values on the page to which the page has been posted.

ADO.NET

153. WHAT IS OBJECT POOLING?

Object pooling is nothing but a repository of the objects in memory which can be used later. This object pooling reduces the load of object creation when it is needed. Whenever there is a need of object, object pool manager will take the request and serve accordingly.

154. WHAT IS CONNECTION POOLING?

Connection pooling consists of database connection so that the connection can be used or reused whenever there is request to the database. This pooling technique enhances the performance of executing the database commands. This pooling definitely reduces our time and effort.

155. WHAT IS THE DIFFERENCE BETWEEN DATAREADER AND DATASET?

- a) Datareader is FORWARD and READ only
- b) Dataset used to UPDATE records.

- a) Datareader is CONNECTED architecture
- b) Dataset is DISCONNECTED Recordset

- a) Datareader contains single table

b) Datareader can contains multiple tables.

- a) Datareader Occupies Less Memory
 - b) Dataset Occupies More memory
-

156. WHAT ARE ALL COMPONENTS OF ADO.NET DATA PROVIDER?

Following are the components of ADO.Net Data provider:

- Connection object – Represents connection to the Database
 - Command object – Used to execute stored procedure and command on Database
 - ExecuteNonQuery – Executes command but doesn't return any value
 - ExecuteScalar – Executes and returns single value
 - ExecuteReader – Executes and returns result set
 - DataReader – Forward and read only recordset
 - DataAdapter – This acts as a bridge between database and a dataset.
-

157. IS IT POSSIBLE TO LOAD MULTIPLE TABLES IN A DATASET?

Yes, it is possible to load multiple tables in a single dataset. **29. Which provider is used to connect MS Access, Oracle, etc...?**

OLEDB Provider and ODBC Provider are used to connect to MS Access and Oracle. Oracle Data Provider is also used to connect exclusively for oracle database.

158. WHAT ARE DIFFERENT LAYERS OF ADO.NET?

There are three different layers of ADO.Net:

- Presentation Layer
 - Business Logic Layer
 - Database Access Layer
-

159. WHAT ARE ALL THE CLASSES THAT ARE AVAILABLE IN SYSTEM.DATA NAMESPACE?

Following are the classes that are available in System.Data Namespace:

- Dataset.
 - DataTable.
 - DataColumn.
 - DataRow.
 - DataRelation.
 - Constraint.
-

160. WHAT ARE THE DATA PROVIDERS IN ADO.NET?

Following are the Data Providers used in ADO.Net:.

- MS SQL Server.
 - OLEDB.
 - ODBC.
-

161. WHAT IS THE DIFFERENCE BETWEEN EXECUTESCALAR AND EXECUTENONQUERY?

1. ExecuteScalar returns output value where as ExecuteNonQuery does not return any value but the number of rows affected by the query.
 2. ExecuteScalar used for fetching a single value and ExecuteNonQuery used to execute Insert and Update statements.
-

162. WHICH METHOD IS USED BY COMMAND CLASS TO EXECUTE SQL STATEMENTS THAT RETURN SINGLE VALUE?

ExecuteScalar

SQL

163. WHAT IS DENORMALIZATION.

DeNormalization is a technique used to access the data from higher to lower normal forms of database. It is also process of introducing redundancy into a table by incorporating data from the related tables.

164. WHAT ARE ALL THE DIFFERENT NORMALIZATIONS?

The normal forms can be divided into 5 forms, and they are explained below -.

First Normal Form (1NF):

This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

Second Normal Form (2NF):

Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

Third Normal Form (3NF):

This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.

Fourth Normal Form (4NF):

Meeting all the requirements of third normal form and it should not have multi-valued dependencies.

165. WHAT IS A RELATIONSHIP AND WHAT ARE THEY?

Database Relationship is defined as the connection between the tables in a database. There are various data basing relationships, and they are as follows::

- One to One Relationship.
 - One to Many Relationship.
 - Many to One Relationship.
 - Self-Referencing Relationship.
-

166. WHAT ARE THE TYPES OF SUBQUERY?

There are two types of subquery – Correlated and Non-Correlated.

A correlated subquery cannot be considered as independent query, but it can refer the column in a table listed in the FROM the list of the main query.

A Non-Correlated sub query can be considered as independent query and the output of subquery are substituted in the main query.

167. WHAT ARE LOCAL AND GLOBAL VARIABLES AND THEIR DIFFERENCES?

Local variables are the variables which can be used or exist inside the function. They are not known to the other functions and those variables cannot be referred or used. Variables can be created whenever that function is called.

Global variables are the variables which can be used or exist throughout the program. Same variable declared in global cannot be used in functions. Global variables cannot be created whenever that function is called.

168. WHAT IS DATA INTEGRITY?

Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

169. WHAT IS DATAWAREHOUSE?

Datawarehouse is a central repository of data from multiple sources of information. Those data are consolidated, transformed and made available for the mining and online processing. Warehouse data have a subset of data called Data Marts.

170. WHAT IS CROSS-JOIN?

Cross join defines as Cartesian product where number of rows in the first table multiplied by number of rows in the second table. If suppose, WHERE clause is used in cross join then the query will work like an INNER JOIN.

171. WHAT IS COLLATION?

Collation is defined as set of rules that determine how character data can be sorted and compared. This can be used to compare A and, other language characters and also depends on the width of the characters.

ASCII value can be used to compare these character data.

172. WHAT ARE ALL DIFFERENT TYPES OF COLLATION SENSITIVITY?

Following are different types of collation sensitivity -.

- Case Sensitivity – A and a and B and b.
 - Accent Sensitivity.
 - Kana Sensitivity – Japanese Kana characters.
 - Width Sensitivity – Single byte character and double byte character.
-

173. ADVANTAGES AND DISADVANTAGES OF STORED PROCEDURE?

Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data.

Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

174. WHAT IS ONLINE TRANSACTION PROCESSING (OLTP)?

Online Transaction Processing (OLTP) manages transaction based applications which can be used for data entry, data retrieval and data processing. OLTP makes data management simple and efficient. Unlike OLAP systems goal of OLTP systems is serving real-time transactions.

Example – Bank Transactions on a daily basis.

175. WHAT IS CLAUSE?

SQL clause is defined to limit the result set by providing condition to the query. This usually filters some rows from the whole set of records.

Example – Query that has WHERE condition

Query that has HAVING condition.

176. WHAT IS RECURSIVE STORED PROCEDURE?

A stored procedure which calls by itself until it reaches some boundary condition. This recursive function or procedure helps programmers to use the same set of code any number of times.

177. WHAT IS UNION, MINUS AND INTERACT COMMANDS?

UNION operator is used to combine the results of two tables, and it eliminates duplicate rows from the tables.

MINUS operator is used to return rows from the first query but not from the second query. Matching records of first and second query and other rows from the first query will be displayed as a result set.

INTERSECT operator is used to return rows returned by both the queries.

178. WHAT IS AN ALIAS COMMAND?

ALIAS name can be given to a table or column. This alias name can be referred in WHERE clause to identify the table or column.

Example-.

Select st.StudentID, Ex.Result from student st, Exam as Ex where st.studentID = Ex.StudentID

Here, st refers to alias name for student table and Ex refers to alias name for exam table.

179. HOW CAN YOU CREATE AN EMPTY TABLE FROM AN EXISTING TABLE?

Example will be -.

Select * into studentcopy from student where 1=2

Here, we are copying student table to another table with the same structure with no rows copied.

180. HOW TO FETCH COMMON RECORDS FROM TWO TABLES?

Common records result set can be achieved by -.

Select studentID from student INTERSECT Select StudentID from Exam

181. HOW TO FETCH ALTERNATE RECORDS FROM A TABLE?

Records can be fetched for both Odd and Even row numbers -.

To display even numbers-.

Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=0

To display odd numbers-.

Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=1

from (Select rowno, studentId from student) where mod(rowno,2)=1.[/sql]

182. HOW TO SELECT UNIQUE RECORDS FROM A TABLE?

Select unique records from a table by using DISTINCT keyword.

Select DISTINCT StudentID, StudentName from Student.

183. WHAT IS THE COMMAND USED TO FETCH FIRST 5 CHARACTERS OF THE STRING?

There are many ways to fetch first 5 characters of the string -.

Select SUBSTRING(StudentName,1,5) as studentname from student

Select LEFT(Studentname,5) as studentname from student

184. WHAT ARE ALL TYPES OF USER DEFINED FUNCTIONS?

Three types of user defined functions are.

- Scalar Functions.
 - Inline Table valued functions.
 - Multi statement valued functions.
-

185. WHAT ARE AGGREGATE AND SCALAR FUNCTIONS?

Aggregate functions are used to evaluate mathematical calculation and return single values. This can be calculated from the columns in a table. Scalar functions return a single value based on the input value.

Example -.

Aggregate – max(), count - Calculated with respect to numeric.

Scalar – UCASE(), NOW() – Calculated with respect to strings.

186. WHICH OPERATOR IS USED IN QUERY FOR PATTERN MATCHING?

LIKE operator is used for pattern matching, and it can be used as -.

% - Matches zero or more characters.

_(Underscore) – Matching exactly one character.

Example -.

```
Select * from Student where studentname like 'a%'
```

```
Select * from Student where studentname like 'ami_'
```

187. DEFINE MAGIC TABLES IN SQL SERVER?

A Table which is automatically created and managed by SQL server internally to store the inserted, updated values for any DML (SELECT, DELETE, UPDATE, etc.) operation, is called as Magic tables in SQL server. The triggers preferably use it.

DESIGN PATTERNS

188. WHAT IS PROTOTYPE DESIGN PATTERN?

Prototype Design patterns:

- Prototype pattern specifies the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.
 - It is used to create a duplicate object or clone of the current object to enhance performance.
-

189. WHAT IS BUILDER DESIGN PATTERN?

Builder Design patterns:

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.
 - In other words, you will have to design the system in such a way that the client application will simply specify the parameters that should be used to create the complex object and the builder will take care of building the complex object.
-

190. WHAT IS ADAPTER DESIGN PATTERN?

Adapter Design patterns:

- The adapter pattern is adapting between classes and objects

- This pattern involves a single class called adapter which is responsible for communication between two independent or incompatible interfaces
 - This works like a bridge between two incompatible interfaces
-

191. WHAT IS BRIDGE DESIGN PATTERN?

Bridge Design patterns:

- Bridge Pattern separates abstraction from its implementation, so that both can be modified Independently
 - Bridge Pattern behaves like a bridge between abstraction class and Implementer class.
-

192. WHAT IS COMPOSITE DESIGN PATTERN?

Composite Design patterns:

- Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchies.
 - Composite pattern creates a class contains group of its own objects. This class provides ways to modify its group of same objects.
 - Composite pattern is used when we need to treat a group of objects and a single object in the same way
-

193. WHAT IS DECORATOR DESIGN PATTERN?

Decorator Design patterns:

- ⊕ Decorator pattern is used to add new functionality to an existing object without changing its structure.
 - ⊕ Decorators provide a flexible alternative to subclass for extending functionality.
 - ⊕ This pattern creates a decorator class which wraps the original class and add new behaviors/operations to an object at run-time.
-

194. WHAT IS FACADE DESIGN PATTERN?

Facade Design patterns:

- ⊕ Facade Design Pattern makes a software library easier to use, understand and test
 - ⊕ Facade Design Pattern make the library more readable
 - ⊕ Facade Design Pattern reduce dependencies of outside code on the inner workings of a library
 - ⊕ Facade Design Pattern wrap a poorly designed collection of APIs with a single well-designed API.
-

195. WHAT IS FLYWEIGHT DESIGN PATTERN?

Flyweight Design patterns:

- ⊕ Flyweight design pattern is an object that minimizes memory use by sharing as much data as possible with other similar objects
- ⊕ Flyweight pattern is used to reduce the number of objects created, to decrease memory and resource usage. As a result it increase performance
- ⊕ Flyweight design pattern provides a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory.
- ⊕ The flyweight pattern uses the concepts of intrinsic and extrinsic data. Intrinsic data is held in the properties of the shared flyweight objects. This information is stateless and generally remains unchanged, if any change

occurs it would be reflected among all of the objects that reference the flyweight. Extrinsic data is computed on the fly means at runtime and it is held outside of a flyweight object. Hence it can be stateful.

196. WHAT IS PROXY DESIGN PATTERN?

Proxy Design patterns:

- Proxy Design pattern involves a class, called proxy class, which represents functionality of another class.
 - Proxy is a wrapper or agent object that is being called by the client to access the real serving object behind the scenes.
-

WEB API / WEB SERVICE / WCF

197. IS IT RIGHT THAT ASP.NET WEB API HAS REPLACED WCF?

It's a not at all true that ASP.NET Web API has replaced WCF. In fact, it is another way of building non-SOAP based services, i.e., plain XML or JSON string.

198. WEB API SUPPORTS WHICH PROTOCOL?

Web App supports HTTP protocol.

199. WHICH .NET FRAMEWORK SUPPORTS WEB API?

NET 4.0 and above version supports web API.

200. WEB API USES WHICH OF THE FOLLOWING OPEN-SOURCE LIBRARY FOR JSON SERIALIZATION?

Web API uses Json.NET library for JSON serialization.

201. BY DEFAULT, WEB API SENDS HTTP RESPONSE WITH WHICH OF THE FOLLOWING STATUS CODE FOR ALL UNCAUGHT EXCEPTION?

500 - Internal Server Error

202. WHAT IS SOAP?

SOAP is an XML message format used in web service interactions. It allows to send messages over HTTP or JMS, but other transport protocols can be used. It is also an XML-based messaging protocol for exchanging information among computers.

203. WHAT IS THE BENEFIT OF USING REST IN WEB API?

REST is used to make fewer data transfers between client and server which make it an ideal for using it in mobile apps. Web API also supports HTTP protocol. Therefore, it reintroduces the traditional way of the HTTP verbs for communication.

204. HOW CAN WE USE WEB API WITH ASP.NET WEB FORM?

Web API can be used with ASP.NET Web Form

It can be performed in three simple steps:

1. Create a Web API Controller,
2. Add a routing table to Application_Start method of Global.sax
3. Then you need to make a jQuery AJAX Call to Web API method and get data.

205. HOW YOU CAN RETURN VIEW FROM ASP.NET WEB API METHOD?

No, we can't return a view from ASP.NET Web API Method. Web API creates HTTP services that render raw data. However, it's also possible in ASP.NET MVC application.

206. HOW TO REGISTER EXCEPTION FILTER GLOBALLY?

It is possible to register exception filter globally using following code-

```
GlobalConfiguration.Configuration.Filters.Add(new  
MyTestCustomerStore.NotImplExceptionFilterAttribute());
```

207. HOW CAN YOU RESTRICT ACCESS METHODS TO SPECIFIC HTTP VERBS IN WEB API?

With the help of Attributes (like HTTP verbs), It is possible to implement access restrictions in Web API.

It is possible to define HTTP verbs as an attribute to restrict access. Example:

```
[HttpPost]
```

```
public void Method1(Class obj)
```

```
{
```

```
//logic
```

208. WHO CAN CONSUME WEBAPI?

WebAPI can be consumed by any client which supports HTTP verbs such as GET, PUT, DELETE, POST. As WebAPI services don't need any configuration, they are very easy to consume by any client. In fact, even portable devices like Mobile devices can easily consume WebAPI which is certainly the biggest advantages of this technology.

209. WHAT ARE WEB SERVICES?

Web services are open standard (XML, SOAP, HTTP etc.) based Web applications that interact with other web applications for the purpose of exchanging data. Web Services can convert your existing applications into Web-applications.

210. WHAT ARE THE FEATURES OF WEB SERVICES?

Following are the features of Web service –

- It is available over the Internet or private (intranet) networks.
 - It uses a standardized XML messaging system.
 - It is not tied to any one operating system or programming language.
 - It is self-describing via a common XML grammar.
 - It is discoverable via a simple find mechanism.
-

211. WHAT THE COMPONENTS OF A WEB SERVICE?

The basic web services platform is XML + HTTP. All the standard web services work using the following components –

- SOAP (Simple Object Access Protocol)
 - UDDI (Universal Description, Discovery and Integration)
 - WSDL (Web Services Description Language)
-

212. HOW DOES A WEB SERVICE WORK?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

213. WHAT IS THE PURPOSE OF XML IN A WEB SERVICE?

A web services takes the help of XML to tag the data, format the data.

214. WHAT ARE THE BENEFITS OF WEB SERVICES?

Following are the benefits of using web services –

- Exposing the Existing Function on the network – Web services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.
- Interoperability – Web services allow various applications to talk to each other and share data and services among themselves.

- Standardized Protocol – Web services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack.
 - Low Cost of Communication – Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services.
-

215. WHAT DO YOU MEAN BY INTEROPERABILITY OF WEB SERVICES?

Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services. For example, a VB or .NET application can talk to Java web services and vice versa. Web services are used to make the application platform and technology independent.

216. WHAT DO YOU MEAN BY LOOSELY COUPLED ARCHITECTURE OF WEB SERVICES?

A consumer of a web service is not tied to that web service directly. The web service interface can change over time without compromising the client's ability to interact with the service. A tightly coupled system implies that the client and server logic are closely tied to one another, implying that if one interface changes, the other must be updated. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.

217. WHAT IS HTTP?

HTTP stands for Hyper Text Transfer Protocol. Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XML-RPC or SOAP messages to masquerade as HTTP messages.

218. WHAT IS WSDL?

WSDL is an XML-based language for describing web services and how to access them. WSDL stands for Web Services Description Language.

219. WHAT IS UDDI?

UDDI is an XML-based standard for describing, publishing, and finding web services. UDDI stands for Universal Description, Discovery, and Integration.

220. EXPLAIN WHAT IS WCF?

WCF (Windows Communication Framework) is Microsoft framework to make inter-process communication easier. Through various means, it lets you do the communication like MS messaging Queuing, Services, Remoting and so on. It also allows you talk with other .NET apps, or non-Microsoft technologies (like J2EE).

221. MENTION WHAT ARE THE MAIN COMPONENTS OF WCF?

Main components of WCF are

- Service: The working logic
- Host: The path where the data is saved. E.g., .exe, process, windows service
- Endpoints: The way the service is exposed to the outside world

222. EXPLAIN HOW DOES WCF WORKS?

WCF follows the “Software as a Service” model, where all units of functionality are defined as services. For communication, each point is a portal or connection either with the client or other services. It is a program that exposes a collection of endpoints.

223. EXPLAIN WHAT IS THE DIFFERENCE BETWEEN ASMX WEB SERVICES AND WCF?

The difference between WCF and ASMX or ASP.net web service is that ASMX is designed to send and receive messages using SOAP over HTTP only. While the WCF can exchange messages using any format over any transport protocol.

224. WHAT ARE THE TRANSPORT SCHEMAS DOES WCF SUPPORTS?

It supports

- HTTP
 - TCP
 - Peer network
 - IPC (Inter Process Communication)
 - MSMQ
-

225. MENTION WHAT ARE THE WAYS OF HOSTING A WCF SERVICE?

The ways of hosting a WCF service are

- IIS

- Self-Hosting
 - WAS (Windows Activation Service)
-

226. MENTION THE ADDRESS SYNTAX AND THE DIFFERENT FORMATS OF WCF TRANSPORT SCHEME?

Address syntax of WCF transport scheme is

[transport]:// [machine or domain] [: optional port] format

227. IN WCF WHAT ARE DUPLEX CONTRACTS?

Duplex messaging or call-back is used in WCF to communicate with the client. Over different transport system Duplex messaging in WCF is done like TCP, Named pipe and even HTTP. Collectively this is known as duplex contracts in WCF.

228. MENTION WHAT ARE THE DIFFERENT INSTANCE MODES IN WCF?

To a particular service instance WCF binds an incoming message request, so the available modes are

- Per Call: This instance is created for each call, efficient in terms of memory but need to maintain session
 - Per Session: For a complete session of a user instance are created
 - Single: One instance is created which is shared among all the users and shared among all. In terms of memory it is least efficient.
-

229. EXPLAIN WHAT IS SOA?

SOA (Service Oriented Architectural) is a collection of services that determines how two computing entities will communicate with each other to achieve certain business functionality and also how one entity can work on behalf of another entity.

230. WHAT ARE THE TYPES OF DATA CONTRACTS IN WCF?

There are two types of Data Contracts

- Data Contract: Attribute used to define the class
 - Data Member: Attribute used to define the properties
-

231. WHAT ARE THE THREE TYPES OF TRANSACTION MANAGER WCF SUPPORTS?

The types of the transaction manager that WCF supports are

- Light Weight
 - WS- Atomic Transaction
 - OLE Transaction
-

232. NAME THE NAMESPACE THAT IS USED TO ACCESS WCF SERVICE?

System.ServiceModel is used to access WCF service

233. WHAT IS SOA STANDS FOR?

.NET CORE

234. WHAT IS HOST IN ASP.NET CORE?

Host encapsulates all the resources for the app. On startup, ASP.NET Core application creates the host. The Resources which are encapsulated by the host include:

- HTTP Server implementation
- Dependency Injection
- Configuration
- Logging
- Middleware components

235. DESCRIBE THE GENERIC HOST AND WEB HOST?

The host setup the server, request pipeline and responsible for app startup and lifetime management. There are two hosts:

.NET Generic Host

ASP.NET Core Web Host

.NET Generic Host is recommended and ASP.NET Core template builds a .NET Generic Host on app startup.

ASP.NET Core Web host is only used for backwards compatibility.

```

// Host creation

public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder
CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup();
}

```

236. DESCRIBE THE SERVERS IN ASP.NET CORE?

Server is required to run any application. **ASP.NET Core** provides an in-process HTTP server implementation to run the app. This server implementation listen for HTTP requests and surface them to the application as a set of request features composed into an `HttpContext`.

ASP.NET Core use the **Kestrel** web server by default. ASP.NET Core comes with:

- Default Kestrel web server that's cross platform HTTP server implementation.
 - IIS HTTP Server that's in-process server for IIS.
 - HTTP.sys server that's a Windows-only HTTP server and it's based on the HTTP.sys kernel driver and HTTP Server API.
-

237. HOW CONFIGURATION WORKS IN ASP.NET CORE?

In ASP.NET Core, **Configuration** is implemented using various configuration providers. Configuration data is present in the form of key value pairs that can be read by configuration providers as key value from different configuration sources as below.

- appsettings.json - settings file
- Azure Key Vault
- Environment variables
- In-memory .Net objects
- Command Line Arguments
- Custom Providers

By default apps are configured to read the configuration data from appsettings.json, environment variables, command line arguments etc. While reading the data, values from environment variables override appsettings.json data values. 'CreateDefaultBuilder' method provide default configuration.

238. HOW TO READ VALUES FROM APPSETTINGS.JSON FILE?

You can read values from appsettings.json using below code.

```
class Test{  
    // requires using Microsoft.Extensions.Configuration;  
    private readonly IConfiguration Configuration;  
    public TestModel(IConfiguration configuration)  
    {  
        Configuration = configuration;  
    }  
    // public void ReadValues(){
```

```
var val = Configuration["key"]; // reading direct key values
var name = Configuration["Employee:Name"]; // read complex
values
}
}
```

Default configuration provider first load the values from appsettings.json and then from appsettings.Environment.json file.

Environment specific values override the values from appsettings.json file. In development environment appsettings.Development.json file values override the appsettings.json file values, same apply to production environment.

239. WHAT IS THE OPTIONS PATTERN IN ASP.NET CORE?

Options Pattern allow you to access related configuration settings in Strongly typed way using some classes. When you are accessing the configuration settings with the isolated classes, The app should adhere these two principles.

- ⊕ **Interface Segregation Principle (ISP) or Encapsulation:** The class the depend on the configurations, should depend only on the configuration settings that they use.
 - ⊕ **Separation of Concerns:** Settings for different classes should not be related or dependent on one another.
-

240. HOW TO USE MULTIPLE ENVIRONMENTS IN ASP.NET CORE?

ASP.NET Core use environment variables to configure application behavior based on runtime environment. launchSettings.json file sets ASPNETCORE_ENVIRONMENT to Development on local Machine.

241. HOW ASP.NET CORE SERVE STATIC FILES?

In ASP.NET Core, **Static files** such as CSS, images, JavaScript files, HTML are the served directly to the clients. ASP.NET Core template provides a root folder called `wwwroot` which contains all these static files. `UseStaticFiles()` method inside `Startup.Configure` enables the static files to be served to client. You can serve files outside of this webroot folder by configuring Static File Middleware as following.

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(env.ContentRootPath,
        "MyStaticFiles")), // MyStaticFiles is new folder
    RequestPath = "/StaticFiles" // this is requested
    path by client
});
// now you can use your file as below

// profile.jpg is image inside MyStaticFiles/images folder
```

242. EXPLAIN SESSION AND STATE MANAGEMENT IN ASP.NET CORE?

As we know HTTP is a stateless protocol. HTTP requests are independent and does not retain user values. There are different ways to maintain user state between multiple HTTP requests.

- Cookies
- Session State
- TempData

- Query strings
 - Hidden fields
 - HttpContext.Items
 - Cache
-

243. CAN ASP.NET APPLICATION BE RUN IN DOCKER CONTAINERS?

Yes, you can run an ASP.NET application or .NET Core application in Docker containers.

244. EXPLAIN THE CACHING OR RESPONSE CACHING IN ASP.NET CORE?

Caching significantly improves the performance of an application by reducing the number of calls to actual data source. It also improves the scalability. Response caching is best suited for data that changes infrequently. Caching makes the copy of data and store it instead of generating data from original source.

Response caching headers control the response caching. **ResponseCache** attribute sets these caching headers with additional properties.

245. WHAT IS IN-MEMORY CACHE?

In-memory cache is the simplest way of caching by ASP.NET Core that stores the data in memory on web server.

Apps running on multiple server should ensure that sessions are sticky if they are using in-memory cache. Sticky Sessions responsible to redirect subsequent client requests to same server. In-memory cache can store any object but distributed

cache only stores byte[].

IMemoryCache interface instance in the constructor enables the In-memory caching service via ASP.NET Core dependency Injection.

246. WHAT IS DISTRIBUTED CACHING?

Applications running on multiple servers (Web Farm) should ensure that sessions are sticky. For Non-sticky sessions, cache consistency problems can occur. **Distributed caching** is implemented to avoid cache consistency issues. It offloads the memory to an external process. Distributed caching has certain advantages as below.

- ⊕ Data is consistent across client requests to multiple server
- ⊕ Data keeps alive during server restarts and deployments.
- ⊕ Data does not use local memory

IDistributedCache interface instance from any constructor enable distributed caching service via [Dependency Injection](#).

247. WHAT IS XSRF OR CSRF? HOW TO PREVENT CROSS-SITE REQUEST FORGERY (XSRF/CSRF) ATTACKS IN ASP.NET CORE?

Cross-Site Request Forgery (XSRF/CSRF) is an attack where attacker that acts as a trusted source send some data to a website and perform some action. An attacker is considered a trusted source because it uses the authenticated cookie information stored in browser.

For example a user visits some site 'www.abc.com' then browser performs authentication successfully and stores the user information in cookie and perform some actions, In between user visits some other malicious site 'www.bad-user.com' and this site contains some code to make a request to vulnerable site (www.abc.com). It's called cross site part of CSRF.

How to prevent CSRF?

- ⊕ In ASP.NET Core 2.0 or later FormTaghelper automatically inject the antiforgery tokens into HTML form element.

- ⊕ You can add manually antiforgery token in HTML forms by using `@Html.AntiForgeryToken()` and then you can validate it in controller by `ValidateAntiForgeryToken()` method.
 - ⊕ For more you can visit [Prevent Cross-Site Request Forgery \(XSRF/CSRF\)](#)
-

248. EXPLAIN SESSION AND STATE MANAGEMENT IN ASP.NET CORE.

As we know HTTP is a stateless protocol. HTTP requests are independent and does not retain user values. There are different ways to maintain user state between multiple HTTP requests.

- ⊕ Cookies
 - ⊕ Session State
 - ⊕ TempData
 - ⊕ Query strings
 - ⊕ Hidden fields
-

249. HOW TO ACCESS HTTPCONTEXT IN ASP.NET CORE?

ASP.NET Core apps access `HttpContext` through the `IHttpContextAccessor` interface.

250. EXPLAIN THE CACHING AND RESPONSE CACHING IN ASP.NET CORE.

Caching significantly improves the performance of an application by reducing the number of calls to actual data source. It also improves the scalability.

Three types of caching

- ⊕ IN-MEMORY CACHING – Good for single server
- ⊕ DISTRIBUTED CACHE – Good for web farm
- ⊕ RESPONSECACHE attribute

Response caching is best suited for data that changes infrequently.

Response caching headers control the response caching. ResponseCache attribute sets these caching headers with additional properties.

[ResponseCache(Duration = 60)]

```
public class HomeController : Controller
```

```
{
```

```
    public IActionResult Index()
```

```
{
```

```
    return View(new HomeOutputModel
```

```
{
```

```
        LastUpdated = DateTime.Now
```

```
    );
```

```
}
```

```
}
```

THE END...
BEST OF LUCK FOR
INTERVIEWS