**Questions Java et Java 8 (12)**

🡪 Définir le polymorphisme, comment le met-on en place :

Déf : redéfinir une méthode dans une classe fille pour modifier le comportement

🡪 Différence entre polymorphisme et surcharge :

La surcharge : la méthode dans une même classe avec des paramètres différents

🡪 Citez les règles d’utilisation des classes abstraites :

* Une classe doit être abstraite si une de ses méthodes est abstraite
* Une classe abstraite n’est pas instanciable (on ne peut pas utiliser les constructeurs d’une classe abstraite et donc on ne peut pas créer d’objet ou d’instance de cette classe).
* Une classe qui hérite d’une classe abstraite ne devient concrète que si elle implémente toutes les méthodes abstraites de la classe dont elle hérite.
* Une méthode abstraite n’est pas implémentée, mais doit être implémentée dans les sous-classes non abstraites.
* Une classe abstraite peut contenir des méthodes non abstraites et des déclarations d’attributs ordinaires.

🡪 Qu’est-ce qu’une expression Lambda ?

* C’est une instance de classe anonyme qui implémente une interface fonctionnelle

🡪 Qu’est-ce qu’un Stream ?

* Un ensemble d’opérations appliqué à une source de données que l’on peut classer en 2 catégories
* Intermediate : opérations qui ne s’exécutent pas sans l’opération terminale (instructions)
* Terminal : opération qui lance le Stream (exécution)

🡪 Comment terminer un Stream pour qu’il retourne une collection ?

* collect(Collectors.tomap(cle,valeur))
* toList() à partir de V16

🡪 Comment terminer un Stream pour qu’il retourne une map ?

collect(Collectors.toMap()) ou collect(Collectors.groupingBy(cle,Collectors.counting))

🡪 Sur quelle liste est basée l’implémentation de la hashmap ?

LinkedList

🡪 La méthode get(int) est plus rapide sur un arrayList, sur une linkedList ou c’est la même chose ? Pourquoi ?

ArrayList. Car les linkedList ont des références sur les index précédents et suivants. Donc c’est plus gourmand pour une recherche.

🡪 La méthode add(int, Object) est plus rapide sur un arrayList, sur une linkedList ou c’est la même chose ? Pourquoi ?

LinkedList. Car il connait la référence de l’élément suivant et précédent.

🡪 Quelle est la différence entre Checked/unchecked exception ?

La même diff qu’entre Exception et RuntimeException.

Exception : Doit être déclaré dans le code soit avec un throw ou un try catch

Runtime pas besoin

🡪 Quelle est la différence Error/Exception ?

Exception dans le code

Error dans la JVM

🡪 Qu’est-ce qu’une interface fonctionnelle ?

C’est une interface avec une seule méthode.

🡪 A quoi sert l'opération flatMap du Stream, quelle est la différence avec Map ?

flatMap sert a mettre tout au même niveau.

Combiner une liste de List en une List.

Exemple Stream<List<String>> en Stream<String>

🡪 Que signifie sealed Interface / Class ? (V15)

On ne peut implémenter une interface Sealed que par des classes identifiées (permits)

Ex : public sealed interface Service permits Car, Truck

On ne peut implémenter une classe abstraite Sealed que par des classes identifiées (permits)

Ex : public abstract sealed class Vehicle permits Car, Truck

🡪 Citez les 2 types de mémoire en Java :

Stack, Heap

🡪 Dans quel type de mémoire passe le « Garbage Collector »

Dans la Heap

**Questions Angular (8)**

### 2. What is data binding in Angular?

Data binding is one of the most significant and effective elements for creating communication between the DOM and the component. It makes designing interactive apps easier by reducing the need to worry about data pushing and pulling between the component and the template.

**There are Four types of Data binding in Angular:**

* Property Binding
* Event Binding
* String Interpolation
* Two way data binding

**Property Binding:**One method of data binding is called property binding. In property binding, we connect a DOM element's property to a field that is a declared property in our TypeScript component code. In reality, Angular transforms string interpolation into property binding internally.

**Event Binding:**Using event binding, you may respond to DOM events like button clicks and mouse movements. When a DOM event (such as a click, change, or keyup) occurs, the component's designated method is called.

**String Interpolation:**In order to export data from TypeScript code to an HTML template( view ), String Interpolation is a one way data binding approach. The data from the component is shown to the view using the template expression enclosed in double curly braces. The value of a component property is added by using string interpolation.

### 13. What is two way data binding in Angular?

Data sharing between a component class and its template is referred to as two-way data binding. If you alter data in one area, it will immediately reflate at the other end. This happens instantly and automatically, ensuring that the HTML template and TypeScript code are always up to date. Property binding and event binding are coupled in two-way data binding.

**Example:**

**app.component.ts**

import { Component } from "@angular/core";

@Component({

selector: "app",

templateUrl: "./app.component.html",

})

export class AppComponent {

data = "This is an example component of two way data binding.";

}

**app.component.html**

<input [(ngModel)]="data"  type="text">

<br> <br>

<h2> You entered the data:  {{data}}</h2>

**app.module.ts**

import { NgModule } from "@angular/core";

import { BrowserModule } from "@angular/platform-browser";

import { FormsModule } from "@angular/forms";

import { AppComponent } from "./app.component";

@NgModule({

imports: [BrowserModule, FormsModule],

declarations: [AppComponent],

bootstrap: [AppComponent],

})

export class AppModule {}

### 22. What is Eager and Lazy loading?

* **Loading:**The eager loading technique is the default module-loading strategy. Eager loading feature modules are loaded before the program begins. This is primarily utilized for small-scale applications.
* **Lazy Loading:**Lazy loading loads the feature modules dynamically as needed. This speeds up the application. It is utilized for larger projects where all of the modules are not required at the start.

### 23. What is view encapsulation in Angular?

View encapsulation specifies if the component's template and styles can impact the entire program or vice versa.

**Angular offers three encapsulation methods:**

* **Native:**The component does not inherit styles from the main HTML. Styles defined in this component's @Component decorator are only applicable to this component.
* **Emulated (Default):**The component inherits styles from the main HTML. Styles set in the @Component decorator are only applicable to this component.
* **None:**The component's styles are propagated back to the main HTML and therefore accessible to all components on the page. Be wary of programs that have None and Native components. Styles will be repeated in all components with Native encapsulation if they have No encapsulation.

### 24. What are RxJs in Angular ?

RxJS is an acronym that stands for Reactive Extensions for JavaScript. It is used to enable the use of observables in our JavaScript project, allowing us to do reactive programming. RxJS is utilized in many popular frameworks, including Angular since it allows us to compose our asynchronous or callback-based code into a sequence of operations executed on a data stream that releases values from a publisher to a subscriber. Other programming languages, such as Java and Python, offer packages that allow them to develop reactive programs utilizing observables.

Most of the time, rxJs is used in HTTP calls with angular. Because http streams are asynchronous data, we can subscribe to them and apply filters to them.

**Example:**The following is a simple example of how RxJs can be utilized with HTTP calls.

let  stream1 = httpc.get("https://www.example.com/somedata");

let stream2 = stream1.pipe(filter(x=>x>3));

stream2.subscribe(res=>this.Success(res),res=>this.Error(res));

### 25. Explain string interpolation and property binding in Angular.

* String interpolation and property binding are parts of **data-binding** in Angular.
* Data-binding is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template).
* Data-binding can be done in two ways, **one-way** binding and **two-way** binding.
* In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas, it is possible in two-way binding.
* String interpolation and property binding allow only one-way data binding.
* String interpolation uses the double curly braces **{{ }}** to display data from the component. Angular automatically runs the expression written inside the curly braces, for example, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. Using property binding, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets **[ ]** syntax.

### 26. How are observables different from promises?

The first difference is that an Observable is **lazy** whereas a Promise is **eager**.

| **Promise** | **Observable** |
| --- | --- |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

Consider the following Observable:

const observable = rxjs.Observable.create(observer => {

console.log('Text inside an observable');

observer.next('Hello world!');

observer.complete();

});

console.log('Before subscribing an Observable');

observable.subscribe((message)=> console.log(message));

 When you run the above Observable, you can see messages being displayed in the following order:

Before subscribing an Observable

Text inside an observable

Hello world!

As you can see, observables are lazy. Observable runs only when someone subscribes to them hence, the message “Before subscribing…” is displayed ahead of the message inside the observable.

Now let’s consider a Promise:

const promise = new Promise((resolve, reject) => {

console.log('Text inside promise');

resolve('Hello world!');

});

console.log('Before calling then method on Promise');

greetingPoster.then(message => console.log(message));

Running the above promise, the messages will be displayed in the following order:

Text inside promise

Before calling then method on Promise

Hello world!

As you can see the message inside Promise is displayed first. This means that a promise runs before the **then** method is called. Therefore, promises are **eager**.

The next difference is that Promises are always **asynchronous**. Even when the promise is immediately resolved. Whereas an Observable, can be both **synchronous** and **asynchronous**.

The above example of an observable is the case to show that an observable is synchronous. Let’s see the case where an observable can be asynchronous:

const observable = rxjs.Observable.create(observer => {

setTimeout(()=>{

observer.next('Hello world');

observer.complete();

},3000)

});

console.log('Before calling subscribe on an Observable');

observable.subscribe((data)=> console.log(data));

console.log('After calling subscribe on an Observable');

The messages will be displayed in the following order:

Before calling subscribe on an Observable

After calling subscribe on an Observable

Hello world!

You can see in this case, observable runs asynchronously.

The next difference is that Observables can emit **multiple** values whereas Promises can emit only one value.

The biggest feature of using observables is the use of **operators**. We can use multiple operators on an observable whereas, there is no such feature in a promise.

### 27. Explain the concept of Dependency Injection?

Dependency injection is an application design pattern which is implemented by Angular.

It also forms one of the core concepts of Angular.

**So what is dependency injection in simple terms?**

Let’s break it down, dependencies in angular are nothing but services which have functionality. The functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies into our components and directives.

So basically, we are just making dependencies which are injectable across all components of an application.

Let’s understand how DI (Dependency Injection) works:

Consider the following service, which can be generated using: ng g service test

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class TestService {

importantValue:number = 42;

constructor() { }

returnImportantValue(){

return this.importantValue;

}

}

 As one can notice, we can create injectable dependencies by adding the **@Injectable** decorator to a class.

We inject the above dependency inside the following component:

import { TestService } from './../test.service';

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

export class TestComponent implements OnInit {

value:number;

constructor(private testService:TestService) { }

ngOnInit() {

this.value = this.testService.returnImportantValue();

}

}

 One can see we have imported our TestService at the top of the page. Then, we created an instance inside the constructor of the component and implemented the **returnImportantValue** function of the service.

From the above example, we can observe how angular provides a smooth way to inject dependencies in any component.

### 28. What are pipes in Angular explain with an example?

Pipes are functions that simplify the process of wiring up your JavaScript expressions and transforming them into their desired output. They can be compared to, say, string functions in other programming languages. Pipes also allow you to combine multiple expressions together, whether they're all values or some values and some declarations.

**For example:**

var css = myTheme.color | "red" ;

This line would assign a value to css , and it's equivalent to writing out the following code:

Pipes have several built-in functions that allow you to transform data, such as value and extract. We can also create our own custom pipes.

Pipes are data transformers that execute on an Angular Component's output. They take in data and return transformed data. For example, if you have an expression such as number | 1000, the number pipe will take data from the output and transform it into 1000. In Angular, there are many built-in pipes that you can use. You can also create your own custom pipes by implementing the PipeTransform interface in a class.

Pipes receive an input which can be a value expression, a function returning an expression, or even a component property., that outputs a number with a value of 1,000. With a pipe, you can transform this output into a formatted string of "1,000" or "1.000".

**Example:**

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

@Component({

selector: 'app-root',

template: `{{ title | uppercase}}`,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'this is an example of custom pies in angular';

}

**Output:**

**THIS IS AN EXAMPLE OF CUSTOM PIPES IN ANGULAR**

Docker, Kubernetes