

**Guide de préparation aux entretiens clients**

1. **Préparation technique de base**
2. **Se renseigner sur les design patterns:**

**Design Pattern Créationnel**

* **C’est qui un Singleton, comment le créer ?**
* Le **singleton** est un [patron de conception](https://fr.wikipedia.org/wiki/Patron_de_conception) *(design pattern)* dont l'objectif est de garantir qu'une unique instance d'une classe donné sera créée et ainsi offrir un point d'accès universel à cette instance. Il existe plusieurs techniques pour implémenter ce design pattern.

Citons :

LazyLoading (voir GIT repo) : l'instance du Singleton est automatiquement crée au chargement de la classe par son classloader.

Eager Loading (voir GIT repo) : L'initialisation de l'instance est retardée jusqu'au premier appel de "getInstance()"

* **Définition de Factory**

Le design pattern Factory, ou **Fabrique** est un design pattern permettant de séparer la création d'objets dérivant d'une classe mère de leur utilisation. De ce fait, on a alors la possibilité de créer plusieurs objets issue d'une même classe mère. (Voir GIT Repo)

Factory vs Abstract Factory :

Prenons le cas de la construction d’un ordinateur de type (PC ou serveur) :

Dans le cas de design pattern Factory, vous allez donner les spécifications de l’ordinateur (RAM, GPU, CPU, type, …) à la fabrique et c’est à la fabrique de créer le PC demandé.

Dans le cas de design pattern Abstract Factory, vous allez faire recours à une fabrique spécialisé pour construire l’ordinateur.

**Design Pattern Comportemental**

* **Définition de Proxy**

Un proxy est une classe se substituant à une autre classe. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue[1](https://fr.wikipedia.org/wiki/Proxy_(patron_de_conception)#cite_note-gof-1). L'utilisation de ce proxy ajoute une indirection à l'utilisation de la classe à substituer (voir GIT repo)

* **Définition de Observer**

 Il est utilisé pour envoyer un signal à des modules qui jouent le rôle d'observateurs

* **Définition de Commande**

Il encapsule la notion d'*invocation*. Il permet de séparer complètement le code initiateur de l'action, du code de l'action elle-même**.**

* **Définition L’inversion de contrôle**

-(*inversion of control*, IoC) est un patron d'[architecture](https://fr.wikipedia.org/wiki/Architecture_logicielle) commun à tous les [frameworks](https://fr.wikipedia.org/wiki/Framework) (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du framework ou de la couche logicielle sous-jacente.

-L’inversion de contrôle est un terme générique. Selon le problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'[injection de dépendances](https://fr.wikipedia.org/wiki/Injection_de_d%C3%A9pendances) (*dependency injection*) qui est un [patron de conception](https://fr.wikipedia.org/wiki/Patron_de_conception) permettant, en [programmation orientée objet](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet), de découpler les dépendances entre objets.

-Elle permet de transformer l’exécution séquentielle du code à un principe de délégation qui se base sur l’exécution d’une certaine fonction quand certaine chose/certain évenement arrive.

Par exemple ce flot d’exécution :

print "enter your name"

read name

print "enter your address"

read address

etc...

store in database

pouura être transformer à :

when the user types in field a, store it in NAME

when the user types in field b, store it in ADDRESS

when the user clicks the save button, call StoreInDatabase

* **Définition de l’Injection de dépendance**

est un mécanisme qui permet d'implémenter le principe de l'[inversion de contrôle](https://fr.wikipedia.org/wiki/Inversion_de_contr%C3%B4le).

Il consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant sur une description (fichier de configuration ou métadonnées) ou de manière programmatique(annotation). Ainsi les dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution.

* **Définition du MVC pour le monde J2EE**

**2- Java standard :**

* Qu’est-ce que Java ? Comment le définiriez-vous ?

Java est un langage de programmation moderne développé par **Sun Microsystems** (aujourd'hui racheté par **Oracle**)

* Quel est votre niveau sur Java 8 ? Quelles sont les nouveautés de Java 8 ?

Mon niveau en java 8 est moyen.

-Méthodes par défaut et statique dans l’interface.

Interface **default** methods:

* Elle aide à l’extension d’une interface spécifique par l’ajout d’une méthode sans se soucier d’une potentielle erreur dans les classes qui implémentent cette interface.

Interface **static** methods:

* Elle permet de protéger une méthode d’être re-implémentée/redéfinie dans une classe fils. (Voir code GIT). Elles permettent de supprimer les classes utilitaires par l’ajout directe de la méthode utilitaire à l’interface correspondente

-Références de méthodes :

Les références de méthodes permettent d'offrir une syntaxe simplifiée pour invoquer une méthode comme une expression lambda : elles offrent un raccourci syntaxique pour créer une expression lambda dont le but est d'invoquer une méthode ou un constructeur. (Voir GIT)

-Expression lambda.

-Annotation de type.

L’apparition de nouveaux API :

-Date and Time API (package java.time au lieu de java.util.date et java.util.calendar)

Limitation de java.util.date :

-Elle ne supporte pas la notion de timezone.

-Elle est mutable (risque d’erreur et thread insafety).

-Les dates antérieurs à 1900 ne sont pas supportés.

Solution :

L’utilisation de java.time : cette classe est immutable donc threadSafe.

Date (LocalDate), time, date/time (LocalDateTime), time zones, instants, duration (difference between two instants), Period (difference between two dates).

-Interface fonctionnelle :

Ce concept, introduit depuis Java 8, permet de définir une interface n'ayant qu'une et une seule méthode abstraite : c'est grâce à cette restriction qu'il sera possible d'utiliser les lambdas car, lors de l'exécution, Java pourra automatiquement déterminer quelle est la signature de la méthode que le lambda remplace et tout sera automatique.

On pourra annoter cette interface avec @FunctionalInterface.

Il y a plusieurs interfaces fonctionnelles citons par exemple l’interface Runnable.

-Améliorations de la gestion de concurrence.

-Le moteur Javascript Nashorn.

* Quelles sont les différences avec Java 7 ? Quelles sont les nouveautés sur Java 9 ?

**Java 7 vs other java:** multi catch, switch with string, try with resources, **Underscore with variable, Type Inference for Generic Instance, java.nio.file package.**

**Java 8 vs java 7:**

@functionalInterface should be used on interface which have only one 1 abstract method.– Lambda expression is a way to represent method implementation of functional interface.

**Collection API improvements:  
-Collection default method removeIf  
–Spliteterator– used to split or traverse traverse elements sequentially or parallel.  
-Map replaceAll, compute, merge(permet la m.a.j d’un map d’une facon dynamique à l’aide d’une expression lambda en spécifiant le clé à mettre à jour et ainsi la valeur y et l’expression lamda pour la mise à jour methods.**

**Concurrency API improvements:  
ConcurrentHashmap(java 1.6)- compute**(), forEach(), forEachEntry(), forEachKey(), forEachValue(), merge(), reduce() and search() methods.

**Java 9 vs java 8**

**1- Interface private methods** –  
so default methods can share some methods.

**2- Collection factory methods –**  
ex- li.listOf(“a”, “b”, “c”);  
setOf  
mapOf.

**Java 10 vs java 9**

**1- Variable type inference-**so

var name= “kuldeep”;

* Qu’est-ce qu’une expression lambda ? Concernant les « LAMBDA » ; où peut-on les utiliser et à quoi servent telles

?

Lambda permet de représenter une functional interface (une interface qui possède une seule méthode abstraite). Lambda expression implémente cette unique méthode abstraite

* Qu’est-ce qu’un Stream ?

Jusqu’à java 7, effectuer des traitements sur des Collections ou des tableaux Java passait essentiellement par l’utilisation du pattern Iterator. Java 8 nous propose l’API Stream pour simplifier ces traitements en introduisant un nouvel objet, Stream.

Celle-ci tire profit de la nouvelle syntaxe des lambdas expressions pour réduire notre code un maximum tout en améliorant nos performances.

* C’est quoi la différence entre Stream et List

List permet de stocker de stocker des données alors stream permet de manipuler et faire des calculs sur ces données.

List est finie alors que stream est infinie en terme de nombre des éléments à manipuler.

Stream est chargé en mode lazy contrairement à list qui est eager.

List peut être parcouru/consommé plusieurs fois. Contrairement à un Stream.

* Dans un stream, si vous appelez uniquement des méthodes intermédiaires sans appeler des méthodes terminales, quand on lance le programme, que ce passe-t-il ?

Les méthodes intermédiaires retournent un autre stream. Par example, les méthodes filter, map, flatmap. Elles ne seront exécutés que si on appelle une méthode terminale tels que collect.

* Utilisation des « stream», imaginez que vous faîtes une collecte dessus ? Pouvez-vous le réutiliser ?

Non

* Utilisation du « Classpath », imaginez que vous avez une application, sur le

« classpath » de cette application vous avez 2 versions d’une même librairie

(avec les mêmes classes mais pas les mêmes implémentations), peut-on savoir si c’est les classes de la librairie 1 ou 2 qui sont utilisées ?

L’ordre de la classe évoqué dans le paramètre classpath de la commande java est important. Il charge la première classe évoqué dans le classpath.

* Différence entre type primitif et objet ? Différence entre Int et Integer ?

Objet est un type référentiel donc immutable. Il occupe plus de mémoire et ainsi les opérations de conversion d’un type objet à un type primitif et vise vers ça (auto(boxing) et (auto(unboxing)) est couteuse en terme de performance.

En utilise le type primitif si on voudra avoir une valeur par défaut dans ce type ou si on veut qu’elle ne doit jamais égale à nulle ou si on fait beaucoup de calcul.

On utilise le type objet dans les autres cas ou dans le cas on est en train de faire de calcul dans une liste.

Int est un type primitif alors que Integer est la classe wrapper donc un type objet.

Integer est une classe donc contient plusieurs méthodes qui permet par exemple la conversion d’une chaine de caractères à une écriture binaire/hexadécimale...

Une écriture binaire

* Avez-vous utilisé des « moc » (bouchons) ? Exemple j’ai un service qui dépend d’un DAO, comment faire pour monter un service sans attaquer la

BDD ?

* Comment définiriez-vous le type optional ?
* Classe abstraite : définition et utilité
* Différence entre Set, List, Map, Tree, Queue

List est une collection ordonnée, indexée et peut contenir des éléments dupliqués contrairement à Set.

Map c’est une classe clé/valeur qui se base sur le hashing.

**Queue**<E> représente une collection gérée comme une file d'attente(FIFO).

* Différence entre ConcurrentHashMap et HashMap :

[HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) is the Class which is under Traditional Collection and ConcurrentHashMap is a Class which is under Concurrent Collections, apart from this there are various differences between them which are:

* HashMap is non-Synchronized in nature i.e. HashMap is not Thread-safe whereas ConcurrentHashMap is Thread-safe in nature.
* HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously. However, ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.
* While one thread is Iterating the HashMap object, if other thread try to add/modify the contents of Object then we will get Run-time exception saying **ConcurrentModificationException**.Whereas In ConcurrentHashMap we won’t get any exception while performing any modification at the time of Iteration.
* Connaitre les Array, Hashtable, Hashmap, Hashset, Treemap, Collection...

Array posséde une propriété importante pour le hashing, tu peux utiliser directement l’index pour accéder à un élément dans un temps contant.

* Différence entre LinkedList et ArrayList

ArrayList implémente un tableau resizable (sa taille augumente dynamiquement 50%), les opérations get et set sont plus rapide.

LinkedList implémente une double Linked List. Add et remove sont moins couteuse. Il est conseillé d’utiliser LinkedList dans le cas où les opérations ajout/suppression sont plus énormes et importantes que get(random access)/set.

* Différence entre Hashtable, Hashmap, TreeMap

Les trois sont des collections clé valeur.

Le choix de l’index se fait selon la formule suivante : Hash(key)->index

On pourra avoir une collision des index. On va recourir à une LinkedList dans ce cas.

Si une collection utilise comme clé une classe, la classe doit obligatoirement redéfinir les méthodes hashcode et equals.

Hashtable vs Hashmap: hashtable est synchronized (thread safe) contrairement à hashmap

TreeMap est une collection clé/valeur ordonnée en se basant sur la clé



1/6



* Différence entre ArrayList et Vector

Synchronization: La collection Vector est synchronisé (thread safe) contrairement à ArrayList.

**Resize :** A une opération de redimensionnement, vector double sa taille contrairement à arrayList qui ajoute la moitié de la taille à sa taille actuelle.

**Performance:** Vector est moins performante qu’un ArrayList parcequ’elle est protégé de l’accés concurrentiel

* Comment implémenter une Hashtable

// Creating an empty Hashtable

Hashtable<Integer, String> hash\_table =

new Hashtable<Integer, String>();

// Mapping string values to int keys

hash\_table.put(10, "Geeks");

hash\_table.put(15, "4");

* Comment parcourir les éléments d’une Map

On utilise la méthode keyset pour récupérer la liste de clé. Ainsi, on utilise une boucle for pour parcourir la liste de clé et récupérer les valeurs correspondantes.

* Peut-on supprimer un élément d’une Collection dans une boucle for ?

On va avoir l’exception **ConcurrentModificationException.** Car deux opérations sont exécutées au même temps(l’itération et l’opération de suppression)

* Comment remédier à ce problème (Supprimer un élément dans une boucle)

?

Il y a plusieurs methods:

1-Créer une copie de la liste et exécuter l’opérations de suppressions sur la copie (copyList = new ArrayList(list) ;)

2-Utiliser Iterator ou listIterator pour le parcours (while(iterator.hasnext()))

et iterator.remove pour la suppression.

3-Utiliser la nouvelle méthode removeif du java 8 (java.util.Collection) avec des predicates. studentList.removeIf(s -> s.getResult().equals("Fail"));

* Signification de synchronized, volatile, transient

-Un champ transient va être exclue de l’opération de serialization.

-Un champ volatile va être manipulé directement à partir de la mémoire au lieu d’une cached value. Les opérations d’écriture sont plus prioritaires que les opérations de lecture (Il fonctionne selon le principe *happens-before* relationship).

-Un champ/méthode/block synchronized est protégés de l’accés concurrentiel grâce à des monitors (l’accés est protégé grâce à un lock et ainsi un seul thread possédant le lock pourra accéder à la variable/méthode/block).

* Création des threads, thread safe method

-Pour la création de thread: -on peut utiliser la classe père Thread (classe anonyme qui hérite de cette classe ou bien créer une classe fils et redéfinir la méthode run).

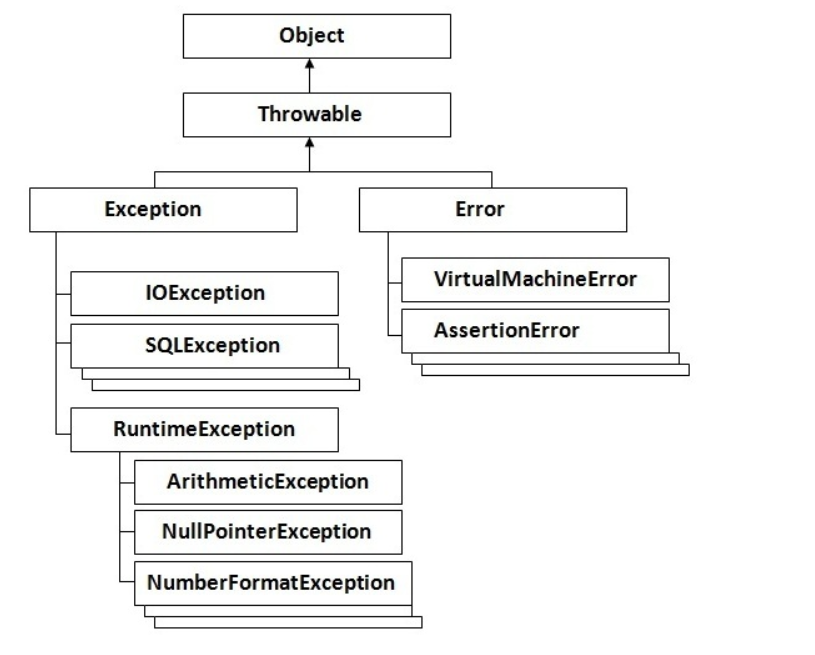
-on peut aussi utiliser l’interface fonctionnel runnable par l’implémentation de cette interface ou la création d’une classe anonyme qui implémente cette interface ou par le biais d’une expression lambda.

-Pour créer une méthode thread safe, on devra utiliser le mot clé synchronized. Si la méthode n’est pas statique, la synchronization est assuré au niveau d’une instance de la classe. On pourra aussi utiliser des variables appartenant à java.util.concurrent.atomic dans la méthode.

* Différents classes d’exception/error/runtime

Checked vs unchecked exception: Checked exceptions sont des exceptions capturées au moment de la compilation alors que les checked exception sont capturées à l’éxecution(Ils héritent de la classe RuntimeException).

Error sont des erreurs fatales du programme non contrôlées par le programmeur (try catch) tels que outofmemory exception.



* Méthodes hashCode() et equals() (définition de chaque une et pour quelle utilisation)

-Equals permet de comparer l’égalité de deux objets(Le premier est l’objet invoquant de la méthodes et le deuxième est passé en paramétre)

Equals doit respecter les contrats suivants :

reflexivité(o1.equals(o1)),

symetrie(o1.equals(o2) => o2.equals(o1)),

transitivité(o1.equals(o2), o2.equals(o3) => o1.equals(o3)),

consistence (o1.equals(o2) quelque soit le temps d’exécution t),

nullcomparision (o1.equals(null) doit toujours retourner false),

Hash code value (o1.equals(o2) => o1.hashcode() = o2.hashcode())

-Hashcode: cette méthode retourne un hash sous forme d’un entier (Id d’un objet). Elle est utilisée dans le hashing (calcul de hash values pour les collections (hashset, hashmap). Si on rédéfinie la methode equals la méthode equals doit être redéfinie aussi. Elle doit respecter les propriétés suivantes :

Consistence (o1.hashcode() ne change pas quelque soit l’instant t)

Hashcode value (o1.equals(o2) => o1.hashcode()= o2.hashcode(), aussi o1.notequals(o2) => on ne peut pas déduire sur l’égalité de hashcode)

* Deux objets égaux ont-ils forcément le même hashcode ?

oui

* Deux objets avec le même hashcode sont-ils forcement égaux ?

non

* Définition du Polymorphisme, les différentes formes du polymorphisme

Le polymorphisme veut dire qu’une méthode/Service peut avoir plusieurs formes ou comportements selon la situation. Il existe deux types de polymorphismes : Runtime(overloading, surcharge) et compileTime(overriding, redéfinition).

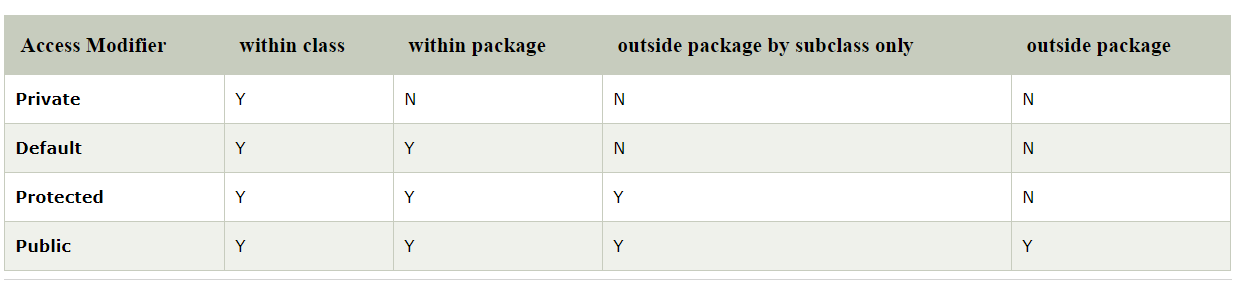
Overriding permet de redéfinir une méthode existante dans une classe mére dans les classes fils.

Overloading : c’est utilisé le même nom à des méthodes ayant des signatures différentes (valeur retourné et paramétres)

* Surcharge et redéfinition en java

Voir la réponse ci-haut.

* Les 4 niveaux d'accessibilité



* Définition d’une classe anonyme

C’est une classe interne dans une classe. Sa déclaration fusionne avec son instanciation. Elle ne posséde pas un nom et elle peut créer q’un seul objet

* Définition des classes internes

Une **classe interne** est déclarée à l'intérieur d'une autre **classe**. Elle peut donc accéder aux membres de la **classe** externe

* Les génériques : dans quel cas les uitiliser

Les génériques (de l'anglais *generics*) sont des classes qui sont typés au moment de la compilation. Autrement dit, ce sont des classes qui utilisent des typages en paramètres. Ainsi une liste chainée, qui peut contenir des entiers, des chaines ou autres, pourra être typée en liste de chaines ou liste d'entiers, et ceci permettra au programmeur de ne pas écrire systématiquement des transtypages(conversion de type), méthode qui pourrait s'avérer dangereuse, ce sera le compilateur qui vérifiera la cohérence des données. Il est introduit avec le java 5.

* La réflexion (reflection) : définition, utilité

La réflexivité, aussi appelée introspection, consiste à découvrir de façon dynamique des informations relatives à une classe ou à un objet. C'est notamment utilisé au niveau de la machine virtuelle Java lors de l'exécution du programme. En gros, la machine virtuelle stocke les informations relatives à une classe dans un objet.

La réflexivité n'est que le moyen de connaître toutes les informations concernant une classe donnée. Vous pourrez même créer des instances de classe de façon dynamique grâce à cette notion.

* Comment fonctionne le garbage collector ? Connaissez-vous une méthode qui permette de déclencher un garbage collector ?

Le garbage collector(ramasse miettes) permet de supprimer/libérer les objets non utilisés/référencés de la mémoire.

Un objet est considéré non référencés s’il est égal à une valeur nulle ou bien il pointe vers une autre référence ou s’il est anonyme.

Si l’objet est libéré, la méthode finalize de l’objet est appelé.

* Peut-on forcer le passage du garbage collector ?

System.gc()

* Multithreading en Java et comment créer un Thread
* Principe de fonctionnement Garbage Collector
* Paramétrage de la mémoire

On distingue principalement deux types de mémoires: la mémoire statique (stack) et la mémoire dynamique (heap).

-La mémoire statique contient des types primitifs/références d’objet stocké dans le heap spécifique à une méthode.

Elle a les spécifications suivantes :

L’accès à cette mémoire est en mode LIFO.

Sa taille pourra être modifiés quand une nouvelle méthode est appelée.

Les variables au sein de cette mémoire sont libérées quand la méthode a fini son exécution.

Si la mémoire est remplie, l’exception *java.lang.StackOverFlowError* est générée.

Le temps d’accès à cette mémoire est beaucoup plus rapide que le heap.

Comme chaque méthode posséde sa propre zone mémoire ou stack, l’accès à cette mémoire est threadSafe.

-La mémoire dynamique ou heap est utilisé pour l’allocation des objets du programme et aussi les classes JRE(Java runtime environment). Cette mémoire pourra être divisée en trois compartiments :

-Young generation : elle contient des objets qui sont récemment alloué. Un garbage collection non couteux en terme de resource va être exécuté quand cette mémoire est remplie.

-Old ou Tenured Generation : Elle contient les objets qui survit depuis longtemps dans la mémoire. Quand un objet résidant dans le compartiment Young generation atteint un certain seuil, il va être déplacer automatiquement à la zone old generation.

-Permanent generation : elle contient les JRE(Java runtime environment).

La mémoire dynamique a les spécifications suivantes :

-Quand la mémoire est remplie, l’exception *java.lang.OutOfMemoryError.*

-La mémoire n’est pas thread safe, elle est aussi plus lente que la mémoire statique et elle devra être libérée par le garbage collector.

-Xms indique la quantité de mémoire utilisée par la heap lors du démarrage de la machine, alors que le paramètre -Xmx est utilisé pour spécifier la quantité de mémoire maximum que peut utiliser la heap lors de l'exécution de la machine

**3- Spring :**

* Qu’est-ce que Spring ? A quoi sert-il ? Quels sont les avantages de l’utilisation de Spring ? En quoi cela peut vous faire gagner du temps ?

Spring Framework est une plateforme open source java. Les fonctionalités de spring permet de développer des applications au-dessus de la plateforme java EE. Spring permet de faciliter grâce à : -Se base sur un modèle **POJO** : Vous n’avez pas besoin d’utiliser un conteneur EJB (un serveur applicatif) pour développer des applications java EE. Vous pourriez uniquement utiliser un conteneur servlet comme par exemple tomcat.

-Spring est organisé d’une façon **modulaire**. Ils existent une multitude de modules et vous pourriez ainsi choisir les modules à utiliser.

-Spring s’intégre facilement avec d’autres technologies et frameworks tels que ORM frameworks, QUARTZ, logging framework comme log4j.

-Ecrire des test cases pour des applications Spring est beaucoup plus facile grâce à son modèle POJO. Ainsi on pourra utiliser l’injection de dépendances pour injecter des data.

-Spring framework contient le module web-mvc qui est une très bonne alternative à STRUTS.

-Spring framework offre une interface pour la gestion de transactions qui s’étend d’une transaction locale (en utilisant une base de donnée locale un seul DBMS) jusqu’à une transaction globale (en utilisant JTA).

* A quoi sert l’utilisation des annotations dans spring ?

Elle permet de faciliter la configuration des beans d’une maniére décentralisé au lieu d’utiliser un fichier XML centraliser. Le conteneur spring doit être informé de l’utilisation d’annotations pour la réalisation de la configuration grâce à l’annotation <context:annotation-config/> dans le fichier de configuration spring.

* Scope des beans (Définit grâce à l’attribut scope)

Spring IOC container va créer :

Singleton : une seule instance d’objet définie par le bean par IOC container. C’est le scope par défaut. A utiliser dans le cas de stateless bean.

Prototype : Une instance d’objet définie par le bean va être crée chaque fois on invoque ce bean.

Request : Une instance d’objet définie par le bean par requête HTTP. A utiliser dans le cas de statefull bean(valide dans le cas d’une application web-aware).

Session : Une instance d’objet définie par le bean par HTTP session (valide dans le cas d’une application web-aware).

Global session : Une instance d’objet définie par le bean par HTTP global session (valide dans le cas d’une application web-aware).

* IOC (L’inversion de contrôle), fonctionnement et but de ce concept.

Voir l’exemple ci-haut.

* Différence entre Spring et les EJBs. Conteneur léger vs lourd.

EJB(Entreprise Java Beans)

Cette architecture propose un cadre pour créer des composants distribués (c’est-à-dire déployés sur des serveurs distants) écrit en langage de programmation [Java](https://fr.wikipedia.org/wiki/Java_(langage)) hébergés au sein d'un [serveur applicatif](https://fr.wikipedia.org/wiki/Serveur_d%27applications) permettant de représenter des données (EJB dit *entité*), , de proposer des services avec ou sans conservation d'état entre les appels (EJB dit *session*), ou encore d'accomplir des tâches de manière asynchrone (EJB dit *message*). Tous les EJB peuvent évoluer dans un contexte [transactionnel](https://fr.wikipedia.org/wiki/Transaction_informatique).

C'est le serveur applicatif(tels que weblogic, JBOSS) qui est chargé de la création, la destruction, la passivation ou l'activation de ses composants en fonction des besoins. Le développer n’a pas besoin de focaliser sur les aspects offerts par l’EJB tels que la sécurité ou bien l’exécution transactionnel. Le client via un appel [RMI](https://fr.wikipedia.org/wiki/Remote_method_invocation_(Java)) (ou une de ses dérivées) va rechercher un EJB par son nom logique [JNDI](https://fr.wikipedia.org/wiki/Java_Naming_and_Directory_Interface) et appeler une ou plusieurs [méthodes](https://fr.wikipedia.org/wiki/M%C3%A9thode_(informatique)) de cet objet.

Spring

EJB vs Spring :

EJB est une spécification JEE alors que SPRING est une implémentation. Ils supportent les deux beaucoup de fonctionnalités comme l’injection de dépendences.

Transaction management : Dans le cas d’un EJB(qui est géré par un conteneur) seulement JTA(transaction globale) est supporté ce qui n’est pas le cas pour Spring qui supporte plusieurs formes de transactions (JTA, JDBC, Hibernate).

Persistence: EJB est fermement intégré avec JPA ce qui n’est pas le cas pour spring qui supporte (JPA, Hibernate, JDBC).

AOP : est plus supporté en SPRING grâce à Spring Aspect et Aspectj.

Conteneur léger vs lourd :

Spring ne demande pas des resources externes comme thirdparty services ou serveur pour s’éxécuter et aussi le code développé par l’application ne dépend pas de conteneur. Donc c’est un conteneur léger ce qui n’est pas le cas pour EJB.

* Connaitre les différents modules et principe de fonctionnement.
* Qu’est-ce que l’injection des dépendances ?
* Les annotations REST de Spring MVC et leur différence avec Jersey
* Quel est le grand concept de Spring ? Le concept qui permet d’utiliser des

@(injection des dépendances)

* C’est quoi le Spring boot ?
* A-t-il une façon de gérer Spring MVC au niveau des contrôleurs ?
* Avec spring mvc, je peux renvoyer de l’XML et comment ?
* Comment configurer Spring ?

2/6



* Que sont les transactions ? Comment utilisez-vous les transactions ? Si on lance une exception, que ce passe-t-il ? Ceci est-il pareil pour tous les cas d’exception ou seulement pour un cas particulier d’exception ?
* Qu’est-ce qu’un attribut transactionnel (ex : tx required) ?
* Comment et quand annuler une transaction ?
* Que sont les niveaux d'isolation des transactions ?
* Qu’est-ce que la notion de profil dans Spring ?
* Lorsque vous annotez votre méthode avec @transactionnal, qu’est-ce que spring fait ? Quel est le mécanisme derrière ?
* Terme utilisé pour citer ce type de manipulation ? (proxy)
* @Autowired vs @Inject vs @Resource
* L’annotation @Qualifier
* Les différents modes de auto wiring
* C’est quoi un inner bean
* Peut-on injecter null et String vide dans Spring ?
* AOP fait-il partie de Spring core ?
* Quelle classe on peut utiliser pour exécuter des requêtes SQL dans Spring ?
* C’est quoi Thread scope
* Le rôle de DispatcherServlet dans Spring MVC
* Spring Data par rapport à Hibernate

**4- Hibernate :**

* Avantages d’un ORM sur JDBC
* Connaitre les composants/Objets : Configuration, SessionFactory, Session, Transaction, Query et Criteria
* SessionFactory est-il thread safe ?
* Session est-il thread safe ?
* Différences entre save() et persist()
* Différences entre get() et load()
* La signification des concepts clés : ORM, cardinalité, proxy, gestion des modifications, identity, les maping one to many…
* Lazyloading
* Les 3 états d’une instance d’une entité
* Différence avec JPA
* Niveaux caches hibernate ? Niveau 1 vs niveau 2 ? Comment configurer le niveau 2 de cache ? différentes stratégies utilisées
* Comment fonctionne le cache de requêtes

**5- Webservices :**

* Définition WSDL : contrat webservices…
* Les implémentations de moteurs SOAP avec Java
* Comment consommer une réponse SOAP ?
* Connaitre quelques outils de génération des classes webservices
* Connaitre le principe des webservices Rest
* RestFul : lister les méthodes
* Microservices : définition



3/6



**6- Angular :**

* Angular JS : directives
* Angular JS vs Angular 2/4/5
* Comment boucler: itérer sur une liste avec typescript: For vs FOR OF vs FOR IN vs FOREACH
* Différence entre JavaScript vs ECMAScript 6 (ES6) vs TypeScript
* L’utilité des annotations @Input et @Output sur Angular
* Comment partager les données entre les composants
* C’est quoi un Model, un Service, un Composant, un Module sur Angular
* Angular CLI : définition + exemples de commandes possibles
* Reactive programming avec RxJS/ngRx et les Observables

**7- Question sur Javascript :**

* Comment pouvez-vous décrire Javascript ? Est-il orienté objet ?
* Côté Front-End, sur angular.js, quels est la différence entre un contrôleur et un service ?
* Quels sont les composants les plus utilisés sur Angular.js ? Comment fonctionne les web-services ? Les contrôleurs ?
* Qu’est-ce que $Scope ? Peut-on se passer de $Scope pour afficher les données d’un contrôleur dans une vue ?
* Qu’est-ce que les routes dans le contrôleur ? Qu’est-ce que UI-Router ?
* Pourriez-vous me citer des designs patterns que vous connaissez ? Comment fonctionne-t-il ?
* Quand un service est un singleton, quand est-il du contrôleur ? Est-ce également un singleton ?
* En javascript natif, qu’est-ce qu’est la fonction apply ?
* Qu’est-ce que la fonction bind ?
* Maitrisez-vous le développement CSS/HTLM ?
* Pourriez-vous me cité des designs patterns que vous connaissez ? Comment fonctionne-t-il ?
* Avez-vous fait la version 2 Angular ?
* Donnez-moi la différence entre NG model et NG bye ?
* Quelle est la dernière version d’AngularJS que vous avez utilisé ?
* Parlez-moi des directives ?
* Comment avez-vous fait pour isoler ses composants ? comment définir l’implémentation ? Scope de directive ?
* La clé scope : **plusieurs possibilités de valeurs et par défaut une scope à quelle** **valeur.** (scope deux points « accolable » ) ça vous parle ?
* Connaissez-vous ES2015 ? Avez-vous utilisé les modules javascript et pas angular ?
* Connaissez-vous vous le spring Template d’angular ?
* La différence entre double égale et triple égale ?
* Différence double côte / simple côte ? sommes-nous interdit d’en utiliser certaines en javascript ?



4/6



**8- SQL-PL/SQL:**

* Tuning des requêtes : ajout d’index…
* C’est quoi un Curseur dans PL/SQL
* C’est quoi une exception ? Comment la créer ?
* C’est quoi les hints ? ça sert à quoi ? Donner un exemple ou 2.
* Fonctions d’agrégation
* C’est quoi les jointures ? C’est quoi une jointure interne et une jointure externe ? Donner un exemple.

**9- Serveur d’application :**

1. Service JNDI
2. Notion datasources

**10- Outils :**

* Jenkins (rôle) ? job Jenkins ?
* SonarQube (rôle, majeur vs critique vs mineur - règles)
* Dette technique : Définition
* Marathon : rôle
* Comment configure-t-on un job jenkins ?
* Pouvez-vous me citer les principales failles de sécurité ? Pourriez-vous me les expliquer ? Comment s’en protège-t-on ?
* Avez-vous déjà utilisé GIT ? En ligne de commande ou à travers une interface graphique ?
* Quelle commande vous feriez pour créer une branche ?
* Avez-vous travailler avec GIT et SVN ? c’est quoi la différence ?
* Connaissez-vous l’agile ? Le Scrum ? Comment cela fonctionnait dans vos anciennes xp ?

**11- Projets :**

* Connaissez-vous les tests unitaires ? Quels outils avez-vous utilisés ?
* Quel est le principe TDD et BDD ? A quoi correspondent-ils ?
* A quoi correspondent les tests d’intégration ?
* Avez-vous déjà codé des tests par exemple avec MGMock qui tourne dans Karma ?
* Comment vous allez tester une classe qui implémente le design paterne singleton pour vérifier avec un test unitaire que la classe respecte bien le design paterne singleton ?
* Différence entre tests unitaires et tests d’intégration
* Mokito : rôle. Différence entre objet Mock et objet Spy
* Tests Postman
* Tests fonctionnelles automatisés (avec les outils : FITNESSE, CUCUMBER, …)

(Définitions)

* Fonctionnement méthodologie Agile/scrum (Cycle ?)
* TDD (Test-Driven Development): définition, cycle



5/6



* + Maven : définition
  + Maven : comment configurer/ automatiser les tests
  + Quel est votre niveau sur Maven ?
  + Quel est le lifecycle de Maven ?
  + Que fait le plugin assembly de Maven ?
  + Pour définir des profils de build dans Maven ?
  + Quelle est la différence entre la commande mvn install et mvn install : install ?
  + Principe d’Intégration continue : Jenkins, repos livrable, tests unitaires, Sonar…
  + Connaitre quelques outils de bug tracking
  + Git vs SVN
  + Comment invoquer un pluging ?
  + Connaissez-vous l’acronyme SOLID ?

**II.Présentation en entretien (Prioritaire)**

Avant toute chose, le consultant doit maitriser la présentation du CV lors des entretiens et connaitre par cœur le contexte de chaque mission dans l’ordre chronologique.

La présentation du CV doit commencer par la dernière mission réalisée:

1. Présenter le nom du projet et son contexte fonctionnel par rapport au client
2. Présenter le service rendu par l’application
3. Détailler votre contribution au sein du projet: conception, développement, support, résolution de bug, réalisation d’évolution, intégration…
4. Présenter l’architecture technique (3 Tiers, n-Tiers…) en commençant par la couche présentation et en terminant par les couches de persistance et/ou interconnexion avec d’autres modules via des connecteurs…
5. Au fur et à mesure de la présentation de l’architecture, vous citerez les

Frameworks et technos utilisés et qui sont détaillés dans la partie

* + environnement technique ».

**III.Préparation aux tests techniques**

Certains clients exigent le passage de test technique comme prérequis avant tout présentation en entretien.

En plus des éléments cités dans le premier paragraphe, vous pouvez vous connecter sur les sites internet suivants qui proposent des QCM avec réponse autour de différents technologies : JEE, Spring, hibernate, JSF, Angular JS,…

1. <http://www.tutorialspoint.com/questions_and_answers.htm>
2. <http://www.buggybread.com/2013/09/java-online-practice-tests.html>

Test Blanc SCJP : Très utile pour préparer le certificat SCJP

1. <http://javatests.hebergratuit.net/?i=2>



6/6



**IV.** **Réussir son entretien Téléphonique/Physique**

Selon la mission, il vous sera demandé de faire un point téléphonique ou de vous présenter pour passer un entretien physique chez le client final ou chez le partenaire.

Dans tous les cas, il est impératif de:

* Respecter l’heure du rendez-vous et de prévoir les aléas des transports en commun.
* Se présenter en costume/cravate (consultants) ou ensemble tailleur (consultantes).
* Avoir toujours sur soi, un cartable avec bloc note et un stylo pour prendre des notes lors de l’entretien.
  1. **ETRE RASSURANT**
* Au début, laisser parler en premier votre interlocuteur.
* Dès le début, votre interlocuteur doit sentir que vous êtes souriant et posé. Aussi, au téléphone le sourire s'entend au bout du fil. Ne surtout pas être stressé.
* Votre interlocuteur doit sentir qu'il est possible de vous faire confiance
* Montrer que vous êtes compétent, efficace avec un bon relationnel.
* Ne dites pas "je ne sais pas " ou " je ne connais pas" mais "j'ai eu l'occasion de le voir, je peux m'y remettre facilement" ou " je peux être opérationnelle rapidement car je suis à l'aise avec la technique et j'apprends vite" ou " je peux être opérationnelle rapidement car j'ai eu l'occasion de le voir" etc...
* Ne pas être timide ou complexé : vous parlez bien le français donc vous pouvez être à l'aise. Aussi, vous êtes rassurant donc il n'y a pas de raison de ne pas convaincre.
* Dire que vous êtes certifié et que votre certification est récente: que c'est frais dans votre tête que vous êtes à jour.
* Etre souple
  1. **ÉCOUTER**
* Bien écouter son interlocuteur, ne pas lui couper la parole.
* Ne pas hésiter à lui demander de reformuler la question si on n’a pas compris. Car c'est très mauvais de répondre à une question qu'on a mal comprise : c'est perdu d'avance!
  1. **SYNTHÉTIQUE:**
* Ne surtout pas parler de toutes les missions, aller à l'essentiel en parlant juste des missions les plus significatives.



7/6



* Parler très brièvement du projet et parler surtout de vos tâches : qu'est-ce que vous faites? Si on vous coupe la parole, n'oubliez pas de reprendre votre argumentaire là ou vous vous êtes arrêté.
* Parler des technologies, des framework, que vous utilisez et citer les versions
* Ne pas aller dans tous les sens : aller à l'essentiel en orientant votre discours vers un cadre professionnel et non pas privé.
* Ne pas parler des stages ou alors ne pas mettre en avant qu'il s'agit d'un stage mais d'une expérience professionnelle.



8/6



**d. MOTIVE:**

* Vous devez vous "vendre" de façon mesurée : lui donner envie de travailler avec vous
* Poser des questions : vos questions ne doivent pas se focaliser sur le client mais sur le poste. Vous devez montrer un intérêt pour le poste et la mission et non pas sur le "potentiel" du client.
* Conclure en vous montrant intéressé.



9/6