Il existe des myriades de questions d'entretien Java et nous ne pouvons pas toutes les rassembler dans un seul article. Cependant tu trouveras ici quelques questions de test d'entretien Java très courantes, d'après les responsables des RH d'entreprises informatiques.

1. **Décomposez l'instruction public static void main(String args[]).**

C'est une des questions les plus populaires et faciles utilisées pour engager la discussion lors d'un entretien Java.

* + public est un modificateur d'accès. Nous l'utilisons pour spécifier l'accès à cette méthode. Ici, le modificateur est « public », ainsi n'importe quelle classe peut accéder à cette méthode.
  + static. Ce mot-clé Java signifie que nous utilisons cette méthode sans créer d'objet de la classe.
  + void est le type de retour de la méthode. Cela signifie que la méthode ne renvoie aucune valeur.
  + main est le nom de la méthode. La machine virtuelle Java (JVM) traite ce nom comme le point d'entrée de l'application (il doit avoir une signature spécifique). main est la méthode où l'exécution principale se produit.
  + String args[]. Il s'agit du paramètre passé à la méthode main. Ici, nous avons des arguments du type String que notre application Java accepte lorsque nous l'exécutons. Tu peux les saisir dans la console.

1. **Quelle est la différence entre equals() et == ?**

Tout d'abord, == est un opérateur, tandis qu'equals() est une méthode. Nous utilisons l'opérateur == pour la comparaison de références (ou d'adresses), et la méthode equals() pour la comparaison de contenu. Cela signifie que == vérifie si les deux objets pointent vers le même emplacement de mémoire, tandis qu'equals() compare les valeurs contenues dans les objets.

1. **Peut-on exécuter un programme sans la méthode main ?**

Beaucoup de questions de base de test Java sont vraiment faciles. Celle-là en est un bon exemple. Alors, la réponse rapide est : oui, c'est possible. Par exemple, nous pouvons le faire à l'aide d'un bloc static.

Tu peux utiliser un bloc static pour initialiser un membre de données static. Il est exécuté avant la méthode main, lorsque les classes sont chargées.

**class** Example {

**static** {

System.out.println("Le bloc static est appelé");

}

**public** **static** **void** main(String args[]) {

System.out.println("Nous sommes maintenant dans la méthode main");

}

}

**Ce qui est affiché :**

Le bloc **static** est appelé

Nous sommes maintenant dans la méthode main

Et si la méthode main est complètement absente ? Si tu essaies d'exécuter une classe ordinaire sans méthode main du tout, tu obtiendras l'erreur suivante : Main method not found in class Test, please define the main method as: public static void main (String [] args) or a JavaFX application class must extend javafx.application.Application. L'erreur elle-même indique que s'il s'agit d'une application JavaFX et que la classe est héritée de javafx.application.Application, alors c'est possible.

1. **Qu'est-ce qu'un objet immuable ? Peut-on modifier un objet immuable ?**

Tu ne peux pas modifier les objets d'une classe immuable une fois qu'ils sont créés. En d'autres termes, une fois que tu les crées, tu ne peux pas les modifier. Si tu essaies de modifier un objet immuable, tu obtiendras un nouvel objet (clone) et tu modifieras ce clone pendant sa création.

String en est un bon exemple. Cette classe est immuable en Java. Cela signifie que tu ne peux pas modifier l'objet String lui-même ; mais tu peux modifier la référence à un objet.

1. **Combien d'objets sont créés dans le code suivant ?**

Il s'agit d'une question technique d'entretien Java qui peut remplacer la n°4 ci-dessus.

String s1 = "Bonjour";

String s2 = "Bonjour";

String s3 = "Bonjour";

La réponse est « un seul », car Java a un pool de String. Lorsque nous créons un objet String à l'aide de l'opérateur new(), Java crée un nouvel objet dans la mémoire du tas. Si nous utilisons la syntaxe littérale de String, comme dans notre exemple ci-dessus, la classe peut renvoyer un objet du pool de String, si celui-ci existe déjà.

1. **Combien d'objets sont créés dans le code suivant ?**

String s = **new** String("Hello");

Il y a 2 objets créés ici. Un dans le pool de constantes String et l'autre dans le tas.

1. **Quelle est la différence entre les classes String, StringBuilder et StringBuffer en Java ?**

C'est une des principales questions d'entretien Java.

Tout d'abord, String est une classe immuable. Cela signifie que tu ne peux pas modifier son contenu après création. Mais StringBuffer et StringBuilder sont des classes mutables, et tu peux donc les modifier plus tard. Si nous essayons de modifier le contenu d'un objet String, la JVM crée une nouvelle chaîne, laissant celle d'origine non modifiée. C'est pourquoi les performances sont meilleures avec StringBuffer qu'avec String.

La principale différence entre StringBuffer et StringBuilder est que les méthodes de StringBuffer sont synchronisées, alors que celles de StringBuilder non.

1. **Y a-t-il une différence entre un objet String créé sous forme littérale et un créé avec l'opérateur new() ?**

Il y en a une. Si nous créons un objet String avec l'opérateur new(), il apparaît dans le tas et n'est pas ajouté au pool de String. Si nous créons un objet String sous forme littérale, il est créé dans le pool de String, qui existe dans la zone PermGen du tas.

1. **Peut-on remplacer une méthode private ou static en Java ?**

Il s'agit d'une question délicate d'entretien Java pour les nouvelles recrues. Tu ne peux pas réellement remplacer une méthode private ou static en Java.

Tu ne peux pas remplacer les méthodes private, car l'étendue du spécificateur d'accès private est limitée à l'intérieur de la classe. Pour remplacer quelque chose, nous avons besoin d'avoir une classe parent et une classe enfant. Si la méthode d'une superclasse est private, alors une classe d'enfant ne peut pas l'utiliser, et les méthodes de la classe enfant seront traitées comme de nouvelles méthodes (et non comme des remplacements).

Les méthodes static ne peuvent pas non plus être remplacées, car les méthodes static font partie de la classe elle-même. Elles ne font partie d'aucun objet de la classe. Bien sûr, tu peux déclarer la même méthode static avec la même signature dans les classes enfant, mais encore une fois, elles seront traitées comme de nouvelles méthodes.

1. **Quelle est la différence entre une classe abstraite et une interface ?**

Il s'agit d'une question populaire d'entretien pour les postes de développeur Java qui porte sur les principes de la POO. Tout d'abord, en Java, une interface définit un comportement tandis qu'une classe abstraite crée une hiérarchie.

|  |  |
| --- | --- |
| **Classe abstraite** | **Interface** |
| Il est possible d'avoir un corps de méthode (méthodes non abstraites) dans une classe abstraite | Une interface peut seulement avoir des méthodes abstraites. Dans Java 8 et les versions ultérieures, il est devenu possible de définir des méthodes par défaut et de les implémenter directement dans une interface. En outre, les interfaces peuvent avoir des méthodes static depuis Java 8. |
| Des variables d'instance peuvent se trouver dans une classe abstraite | Une interface ne peut pas avoir de variables d'instance. |
| Les constructeurs sont autorisés | Une interface ne peut pas avoir de constructeur. |
| Les méthodes static sont autorisées | Les méthodes static ne sont pas autorisées |
| Une classe ne peut avoir qu'un seul parent abstrait | Une classe peut implémenter plusieurs interfaces |
| Une classe abstraite peut fournir l'implémentation de l'interface. | Une interface ne peut pas fournir l'implémentation d'une classe abstraite. |
| Une classe abstraite peut étendre une autre classe Java et implémenter plusieurs interfaces Java. | Une interface est uniquement autorisée à étendre une autre interface Java. |
| Une classe Java abstraite peut avoir des membres de classe private et protected | Les membres d'une interface Java sont public par défaut |

1. **Peut-on déclarer des variables et méthodes static dans une classe abstraite ?**

Oui, il est possible de déclarer des variables et des méthodes static dans une classe abstraite. Il n'est pas nécessaire de créer un objet pour accéder au contexte statique. Par exemple, nous sommes autorisés à accéder au contexte statique déclaré à l'intérieur d'une classe abstraite en utilisant le nom de la classe abstrait.

1. **Quels types de zones de mémoire sont alloués par la JVM ?**

La **zone de classe** stocke des structures pour chaque classe, par exemple un pool de constantes d'exécution, des champs, des données de méthode et tout le code des méthodes.

Le **tas** est une zone de données d'exécution où la mémoire est allouée aux objets.

La **pile** stocke les trames. Elle contient des variables locales et des résultats intermédiaires, et participe aux appels et retours de méthode. Chaque thread a une pile JVM privée, créée en même temps que le thread. Une nouvelle trame est créée chaque fois qu'une méthode est appelée. Une trame est détruite lorsque son appel de méthode se termine.

Le **registre du compteur ordinal** contient l'adresse de l'instruction de machine virtuelle Java en cours d'exécution.

La **pile de méthodes natives** contient toutes les méthodes natives utilisées dans l'application.

1. **Pourquoi l'héritage multiple n'est-il pas autorisé en Java ?**

Ce serait vraiment compliqué. Imagine trois classes A, B et C, où C hériterait de A et de B. Supposons maintenant que les classes A et B ont la même méthode et que tu l'appelles à partir d'un objet de la classe enfant... Quelle méthode sera appelée ? Celle de la classe A ou celle de la classe B ? Nous avons ici une ambiguïté.

Si tu essaies d'hériter de deux classes, Java générera une erreur au moment de la compilation.

1. **Peut-on surcharger la méthode main() ?**

Bien sûr, nous pouvons avoir de nombreuses méthodes main dans un programme Java en utilisant la surcharge de méthode. Essaie un peu !

1. **Peut-on déclarer un constructeur comme final ?**

Non. Un constructeur ne peut pas être déclaré comme final, car il ne peut pas être hérité. Il ne serait donc pas logique de déclarer un constructeur comme final. Toutefois, si tu essaies quand même, le compilateur Java te renverra une erreur.

1. **Peut-on déclarer une interface comme final ?**

Non, nous ne pouvons pas faire ça. Une interface ne peut pas être final, car elle doit être implémentée par une certaine classe sur la base de sa définition. Par conséquent, il ne serait pas logique de déclarer une interface comme final. Toutefois, si tu essaies de le faire, le compilateur Java te renverra une erreur.

1. **Quelle est la différence entre la liaison statique et la liaison dynamique ?**

Une liaison qui peut être résolue au moment de la compilation par le compilateur est appelée liaison static ou préalable. La liaison de toutes les méthodes static, private et final se fait au moment de la compilation.

Avec la liaison dynamique, le compilateur ne peut pas déterminer la méthode à appeler. Le remplacement de méthode est un exemple parfait de liaison dynamique. Avec le remplacement de méthode, les classes parent et enfant ont la même méthode.

Static Binding

**class** Cat{

**private** **void** talk() {

System.out.println("Le chat miaule...");

}

**public** **static** **void** main(String args[]) {

Cat cat=**new** Cat();

cat.talk();

}

}

Liaison dynamique

**class** Animal {

**void** talk() {

System.out.println("L'animal parle...");

}

}

**class** Cat **extends** Animal {

**void** talk() {

System.out.println("Le chat parle...");

}

**public** **static** **void** main(String args[]) {

Animal animal=**new** Cat();

animal.talk();

}

}

1. **Comment créer une classe en lecture seule en Java ?**

Tu peux le faire en rendant tous les champs de la classe private. Une classe en lecture seule n'a que des méthodes getter, qui renvoient les propriétés private de la classe à la méthode main. Tu ne peux pas modifier ces propriétés en raison de l'absence de setters.

**public** **class** HockeyPlayer {

**private** String team ="Feuille d'érable";

**public** String getTeam() {

**return** team;

}

}

1. **Comment créer une classe en écriture seule en Java ?**

Encore une fois, tu dois rendre tous les champs de la classe private. Cette fois, ta classe en écriture seule doit avoir seulement des setters et aucun getter. Ainsi, nous ne pouvons pas lire les propriétés de la classe.

**public** **class** HockeyPlayer {

**private** String team;

**public** **void** setTeam(String college) {

**this**.team = team;

}

}

1. **Chaque bloc try doit être suivi d'un bloc catch. Vrai ou faux ?**

Faux. Ce n'est pas une obligation. Un bloc try ne doit pas nécessairement avoir un bloc catch. Il peut être suivi par un bloc catch, un bloc finally ou aucun des deux.

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**try** {

**int** variable = 1;

System.out.println(variable/0);

}

**finally**

{

System.out.println("Le reste du programme...");

}

}

}

**Sortie :**

Exception in thread main java.lang.ArithmeticException:/ by zero Le reste du programme...

Voici un autre exemple :

**class** Main {

**public** **static** **void** main(String[] args) **throws** IOException {

**try**(InputStreamReader inputStreamReader = **new** InputStreamReader(System.in);

BufferedReader reader = **new** BufferedReader(inputStreamReader)) {

System.out.println("test");

}

}

}

**Sortie :**

test

**P.S. :**Avant Java 8, les méthodes des interfaces pouvaient seulement être abstraites. Dans Java 8 et les versions ultérieures, il est devenu possible de définir des méthodes par défaut et de les implémenter directement dans une interface.

1. **Quelle est la différence entre les mots-clés throw et throws ?**

throws sert à déclarer une exception, et fonctionne donc comme un bloc try-catch. Le mot-clé throw est utilisé pour lever une exception explicitement à partir d'une méthode ou de tout autre bloc de code.

throw est suivi d'une instance de la classe Exception, mais throws est suivi par des noms de classe d'exception.

throw est utilisé dans un corps de méthode pour lever une exception. throws est utilisé dans une signature de méthode pour déclarer les exceptions qui peuvent se produire dans les instructions présentes dans la méthode.

Tu peux lever une seule exception à la fois, mais tu peux gérer plusieurs exceptions en les déclarant à l'aide du mot-clé throws. Tu peux déclarer plusieurs exceptions, p. ex. public void method() throws IOException, SQLException.

Ici, nous n'avons examiné que quelques-unes des questions et réponses courantes d'entretien Java. Cet article est le premier d'une série sur les entretiens. Le prochain (qui arrivera sous peu) portera sur les questions relatives aux structures de données.