**Principes POO :**

* **Encapsulation** : principe de regrouper des données brutes avec un ensemble de [routines](https://fr.wikipedia.org/wiki/Routine_(informatique)) permettant de les lire ou de les manipuler. Chaque [classe](https://fr.wikipedia.org/wiki/Classe_(informatique)) définit des [méthodes](https://fr.wikipedia.org/wiki/M%C3%A9thode_(informatique)) ou des [propriétés](https://fr.wikipedia.org/wiki/Propri%C3%A9t%C3%A9_(informatique)) pour interagir avec les données membres.
* **Héritage** : L’héritage est la définition d’une classe par extension des caractéristiques d’une autre classe, exemple : on a créé une classe *Véhicule*. Ainsi les classes Automobile et Avion ont pu hériter des caractéristiques de *Véhicule*.
* **Polymorphisme** : le même service, aussi appelé opération ou méthode, peut avoir un comportement différent selon les situations.

On peut donner à une même méthode, plusieurs signatures pour implémenter des comportements différents selon les types des paramètres passés. La signature d'une méthode est composée du nom de celle-ci, de sa portée, du type de donnée qu'elle renvoie et enfin du nombre et du type de ses paramètres.

Dans la signature d'une méthode, on peut préciser qu'il est possible de passer plus de 1 paramètre du même type en suffixant le type du paramètre avec « ... ».

* **Interfaces** : Liste de méthodes dont on donne seulement la signature
* Représente un "contrat", ce qu'on attend d'un objet
* Peut être implémentée par une ou plusieurs classes qui doivent donner une implémentation pour chacune des méthodes annoncées (et éventuellement d'autres).
* Une classe peut implémenter plusieurs interfaces (permettant un héritage multiple, en les séparant par des virgules après le mot *implements*).
* Toutes les méthodes d'une interface sont implicitement abstraites.
* Une interface n'a pas de constructeurs
* Une interface ne peut avoir de champs sauf si ceux-ci sont statiques.
* Une interface peut être étendue par une ou plusieurs autre(s) interface(s).

En fait, une interface est une classe abstraite dont toutes les méthodes sont abstraites et dont tous les attributs sont constants (des constantes, voir le mot-clé *final*).

**SOLID** :

* [**Responsabilité unique**](https://fr.wikipedia.org/wiki/Principe_de_responsabilit%C3%A9_unique)**(*single responsibility principle*) :** une [classe](https://fr.wikipedia.org/wiki/Classe_(informatique)), une fonction ou une méthode doit avoir une et une seule responsabilité
* [**Ouvert/fermé**](https://fr.wikipedia.org/wiki/Principe_ouvert/ferm%C3%A9)**(*open/closed principle*) :** une entité applicative (class, fonction, module ...) doit être ouverte à l'extension, mais fermée à la modification
* [**Substitution de Liskov**](https://fr.wikipedia.org/wiki/Principe_de_substitution_de_Liskov)**(*Liskov substitution principle*) :** une instance de type T doit pouvoir être remplacée par une instance de type G, tel que G sous-type de T, sans que cela ne modifie la cohérence du programme
* [**Ségrégation des interfaces**](https://fr.wikipedia.org/wiki/Principe_de_s%C3%A9gr%C3%A9gation_des_interfaces)**(*interface segregation principle*) :** préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale
* [**Inversion des dépendances**](https://fr.wikipedia.org/wiki/Inversion_des_d%C3%A9pendances)**(*dependency inversion principle*) :** il faut dépendre des abstractions, pas des [implémentations](https://fr.wikipedia.org/w/index.php?title=Impl%C3%A9mentations&action=edit&redlink=1)

What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management.

### What is gc()?

The gc() method is used to invoke the garbage collector for cleanup processing. This method is found in System and Runtime classes. This function explicitly makes the Java Virtual Machine free up the space occupied by the unused objects so that it can be utilized or reused. Consider the following example for the better understanding of how the gc() method invoke the garbage collector.

**Les threads :**

Les *threads*, ou fils d'exécution, permettent l'exécution de plusieurs tâches en même temps.

Un processus léger est un contexte d'exécution d'une application. Ce processus possède sa propre pile et pointeur d'exécution.

Une application en cours d'exécution (un processus) peut avoir plusieurs sous-threads. Tous les threads d'un même processus partagent la même zone de données. Ce qui veut dire que toute variable membre d'une classe est modifiable par n'importe quel processus léger. Il faut donc un moyen de synchroniser l'accès aux variables

Un thread possède différents états gérés par le système :

* état *prêt* : le processus est prêt à être exécuté,
* état *suspendu* : le processus est suspendu (attente d'une ressource),
* état *exécution* : le processus est en cours d'exécution,
* état *terminé* : le processus a achevé son exécution ou a été interrompu

La synchronisation devient nécessaire quand plusieurs processus légers accèdent aux mêmes objets.

**mot-clé synchronized**

Le mot-clé synchronized permet un accès exclusif à un objet.

La syntaxe est la suivante :

... code non protégé ...

**synchronized**(objet) {

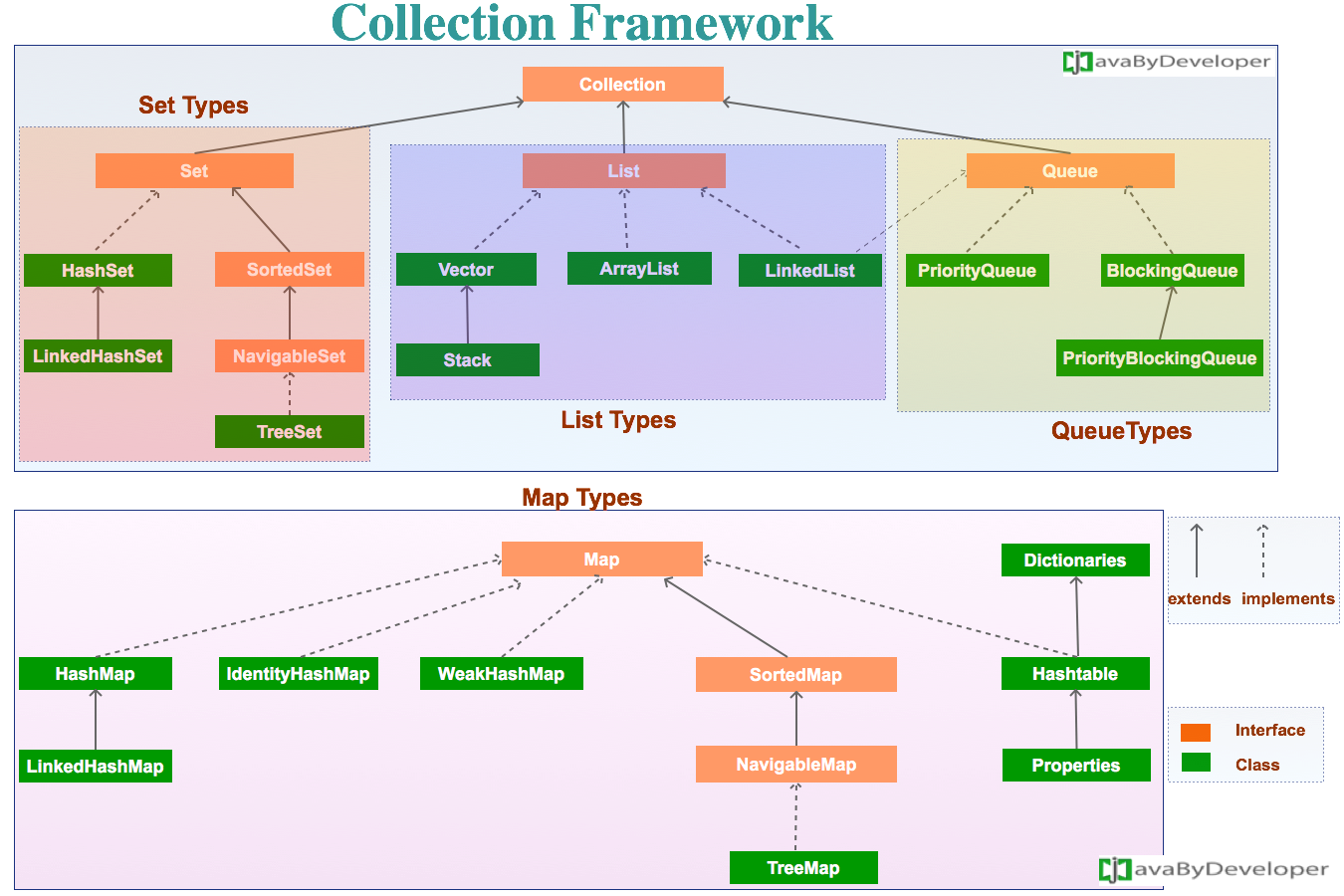
... code protégé ...

}

... code non protégé ...

Le code protégé n'est exécuté que par un seul processus léger à la fois, tant qu'il n'a pas terminé le bloc d'instruction.

**Les Collections**

[](https://secureservercdn.net/160.153.137.59/c78.8cb.myftpupload.com/wp-content/uploads/2016/06/Collection-Framework-hierarchy.png)

**1. Collection Interface :**

1. The root interface in the *collection hierarchy*. A collection represents a group of objects, known as its *elements*.
2. The JDK does not provide any *direct* implementations of this interface: The implementation splitted to its child interfaces and their implementation classes.
3. Collection interface provides the most common general methods which are applicable for any collection object (List, Set, Queue implementations).

**2. List Interface :**

1. An ordered collection (also known as a *sequence*).
2. Insertion order preserved, if user used List type implementation, user has control over the order, because the insertion order and retrieval order of an element is same.
3. If List type implementations used Duplicate elements allowed to insert.

**3. Set Interface :**

1. Insertion order not preserved, if user used Set type implementation, user has no control over the order, because the insertion order and retrieval order of an element is not same.
2. If Set type implementations used Duplicate elements not allowed to insert.

**4. SortedSet Interface :**

1. Child interface of Set. It is a set that holds the elements in natural sorting (Ascending) order.
2. The element which we are adding to a SortedSet implementation that element should *implements*Comparable Interface. Because when we are adding element, it uses Comparable implementation to sort the order. ( Just remember this Interface Comparable, I will explain more in depth details about Comparator and sorting order in further tutorials).

**5. NavigableSet Interface :**

1. It is a set that extends SortedSet, it means it is child Interface of SortedSet.
2. It provides navigational methods. For example if you want to retrieve an element that higher than or less than a given value. If you want such kind of navigation, NavigableSet provides navigational utility methods.

**6. Queue Interface :**

1. A collection designed for holding elements prior to processing.
2. A *queue* supports the insert and remove operations using a FIFO (First-In-First-Out) manner.
3. Queue provides prior to processing methods.

**7. Map Interface :**

1. An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.
2. Map represent each element as key value pair, and that key value pair called Entry.
3. Map interface provides the most common methods that is applicable for any Map implementation type object.

**8. SortedMap Interface :**

1. Child interface of Map. It is a map that holds the key elements in natural sorting (Ascending) order.
2. The key element which we are adding to a SortedMap implementation that element should *implements*Comparable Interface. Because when we are adding element, it uses Comparable implementation to sort the order. ( Just remember this Interface Comparable, I will explain more in depth details about Comparator and sorting order in further tutorials).

**9. NavigableMap Interface :**

1. It is a Map that extends SortedMap, it means it is child Interface of SortedMap.
2. It provides navigational methods. For example if you want to retrieve a key element that higher than or less than a given key. If you want such kind of navigation, NavigableMap provides navigational utility methods.

## **Modificateurs d'accès**

En Java, la déclaration d'une classe, d'une méthode ou d'un membre peut être précédée par un modificateur d'accès.

Un modificateur indique si les autres classes de l'application pourront accéder ou non à la classe/méthode/membre (qualifié par la suite d'« item »).

Ces modificateurs sont au nombre de quatre :

* *public* : toutes les classes peuvent accéder à l'item
* *protected* : seules les classes dérivées et les classes du même [package](https://fr.wikibooks.org/wiki/Programmation_Java/Extensions) peuvent accéder à l'item
* *private* : l'item est seulement accessible depuis l'intérieur de la classe où il est défini.
* (par défaut) : sans modificateur d'accès, seules les classes du même [package](https://fr.wikibooks.org/wiki/Programmation_Java/Extensions) peuvent accéder à l'item.

L'utilisation des modificateurs permet au programmeur de contrôler la visibilité des différents items et permet d'empêcher que des actions illégales soient effectuées sur les items.

## **abstract**

Le modificateur abstract indique qu'une classe ou méthode est [abstraite](https://fr.wikibooks.org/wiki/Programmation_Java/Classes_abstraites).

## **final**

Ajouté devant un attribut, il le rend immuable, dès lors qu'il est initialisé (autrement dit, il n'est pas obligatoire de l'initialiser dès la déclaration, contrairement à d'autres langages). Pour les types primitifs, final fige la valeur, pour les objets, final fige la **référence**, et non la **valeur** de la référence (i.e. seule l'instanciation est figée).

Devant une variable locale (dans une méthode, donc), il a le même comportement que pour un attribut.

Devant une méthode, il indique que cette méthode ne peut pas être modifiée dans une classe dérivée. Les méthodes static et private sont implicitement final.

Devant une classe, il indique que cette classe ne peut pas avoir de sous-classe.

## **static**

Le modificateur static indique, pour une méthode, qu'elle peut être appelée sans instancier sa classe (syntaxe : Classe.methode()).

Pour un attribut, qu'il s'agit d'un attribut de classe, et que sa valeur est donc partagée entre les différentes instances de sa classe.

De plus, il est possible de déclarer dans une classe un bloc d'initialisation statique, qui est un bloc d'instruction précédé du modificateur static.

**Q**uelle est la différence entre **Surcharge**et **Redéfinition**? –

La **surcharge de fonctions** est utilisée dans une seule classe où vous avez le même nom de fonction mais un ensemble d’arguments différent pour chaque fonction.  
   
La **redéfinition de fonction** est un concept que l’on rencontre lors de la création de sous-classes. Ici, si vous déclarez une sous-classe et créez une fonction avec le même nom et les mêmes arguments qu’une fonction dans la classe de base, la fonction associé à la sous-classe sera exécuté. C’est parce que vous allez probablement déclarer un objet de la sous-classe. D’une manière générale, la fonction associée à l’objet créé sera exécutée.

**Les Exceptions**

Une **exception** est un signal qui se déclenche en cas de problème. Les exceptions permettent de gérer les cas d'erreur et de rétablir une situation stable (ce qui veut dire, dans certains cas, quitter l'application proprement). La gestion des exceptions se décompose en deux phases :

* La levée d'exceptions,
* Le traitement d'exceptions.

En Java, une exception est représentée par une classe. Toutes les exceptions dérivent de la classe *Exception* qui dérive de la classe *Throwable*.

## **Levée d'exception**

Une exception est levée grâce à l'instruction *throw* :

**if** (k<0)

**throw** **new** RuntimeException("k négatif");

Une exception peut être traitée directement par la méthode dans laquelle elle est levée, mais elle peut également être envoyée à la méthode appelante grâce à l'instruction *throws* (à ne pas confondre avec *throw*) :

**import** **java.io.IOException**;

**public** void maMethode(int entier) **throws** IOException

{

*//code de la methode*

}

Le traitement des exceptions se fait à l'aide de la séquence d'instructions

*try... catch... finally*.

* L'instruction *try* indique qu'une instruction (ou plus généralement un bloc d'instructions) susceptible de lever des exceptions débute.
* L'instruction *catch* indique le traitement pour un type particulier d'exceptions. Il peut y avoir plusieurs instructions *catch* pour une même instruction *try*.
* L'instruction *finally*, qui est optionnelle, sert à définir un bloc de code à exécuter dans tous les cas, exception levée ou non.

Il faut au moins une instruction *catch* ou *finally* pour chaque instruction *try*.

Exemple :

**public** String lire(String nomDeFichier) **throws** IOException

{

**try**

{

*// La ligne suivante est susceptible de lever une exception*

*// de type FileNoFoundException*

FileReader lecteur = **new** FileReader(nomDeFichier);

char[] buf = **new** char[100];

*// Cette ligne est susceptible de lever une exception*

*// de type IOException*

lecteur.read(buf,0,100);

**return** **new** String(buf);

}

**catch** (FileNotFoundException fnfe)

{

fnfe.printStackTrace(); *// Indique l'exception sur le flux d'erreur standard*

}

**finally**

{

System.err.println("Fin de méthode");

}

}

Le bloc catch (FileNotFoundException fnfe) capture toute exception du type FileNotFoundException (cette classe dérive de la classe IOException).

Le bloc finally est exécuté quel que soit ce qui se passe (exception ou non).

Toute autre exception non capturée (telle IOException) est transmise à la méthode appelante, et doit **toujours** être déclarée pour la méthode, en utilisant le mot clé throws, sauf les exceptions dérivant de la classe RuntimeException.

S'il n'y avait pas cette exception à la règle, il faudrait déclarer throws ArrayIndexOutOfBoundsException chaque fois qu'une méthode utilise un tableau, ou throws ArithmeticException chaque fois qu'une expression est utilisée, par exemple.

**Spring**

**What Are the Differences Between Spring and Spring Boot?**

The Spring Framework provides multiple features that make the development of web applications easier. These features include dependency injection, data binding, aspect-oriented programming, data access, and many more.

Over the years, Spring has been growing more and more complex, and the amount of configuration such application requires can be intimidating. This is where Spring Boot comes in handy – it makes configuring a Spring application a breeze.

Essentially, while Spring is unopinionated, **Spring Boot takes an opinionated view of the platform and libraries, letting us get started quickly.**

Here are two of the most important benefits Spring Boot brings in:

* Auto-configure applications based on the artifacts it finds on the classpath
* Provide non-functional features common to applications in production, such as security or health checks

**How Can We Set up a Spring Boot Application with Maven?**

* We can include Spring Boot in a Maven project just like we would any other library. However, the best way is to inherit from the *spring-boot-starter-parent* project and declare dependencies to [Spring Boot starters](https://www.baeldung.com/spring-boot-starters). Doing this lets our project reuse the default settings of Spring Boot.
* Inheriting the *spring-boot-starter-parent* project is straightforward – we only need to specify a *parent* element in *pom.xml*:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <parent>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-parent</artifactId>      <version>2.1.1.RELEASE</version>  </parent> |

**How to Disable a Specific Auto-Configuration?**

If we want to disable a specific auto-configuration, we can indicate it using the *exclude* attribute of the *@EnableAutoConfiguration* annotation. For instance, this code snippet neutralizes *DataSourceAutoConfiguration*:

|  |  |
| --- | --- |
| 1  2  3 | // other annotations  @EnableAutoConfiguration(exclude = DataSourceAutoConfiguration.class)  public class MyConfiguration { } |

If we enabled auto-configuration with the *@SpringBootApplication* annotation — which has *@EnableAutoConfiguration* as a meta-annotation — we could disable auto-configuration with an attribute of the same name:

|  |  |
| --- | --- |
| 1  2  3 | // other annotations  @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)  public class MyConfiguration { } |

We can also disable an auto-configuration with the *spring.autoconfigure.exclude* environment property. This setting in the *application.properties* file does the same thing as before:

|  |  |
| --- | --- |
| 1 | spring.autoconfigure.exclude=  org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration |

### ****How to Tell an Auto-Configuration to Back Away When a Bean Exists?****

To instruct an auto-configuration class to back off when a bean is already existent, we can use the @ConditionalOnMissingBean annotation. The most noticeable attributes of this annotation are:

* value: The types of beans to be checked
* name: The names of beans to be checked

When placed on a method adorned with @Bean, the target type defaults to the method's return type:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Configuration  public class CustomConfiguration {      @Bean      @ConditionalOnMissingBean      public CustomService service() { ... }  } |

**How to Deploy Spring Boot Web Applications as Jar and War Files?**

Traditionally, we package a web application as a WAR file, then deploy it into an external server. Doing this allows us to arrange multiple applications on the same server. During the time that CPU and memory were scarce, this was a great way to save resources.

However, things have changed. Computer hardware is fairly cheap now, and the attention has turned to server configuration. A small mistake in configuring the server during deployment may lead to catastrophic consequences.

Spring tackles this problem by providing a plugin, namely spring-boot-maven-plugin, to package a web application as an executable JAR. To include this plugin, just add a plugin element to pom.xml:

|  |  |
| --- | --- |
| 1  2  3  4 | <plugin>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-maven-plugin</artifactId>  </plugin> |

With this plugin in place, we'll get a fat JAR after executing the package phase. This JAR contains all the necessary dependencies, including an embedded server. Thus, we no longer need to worry about configuring an external server.

We can then run the application just like we would an ordinary executable JAR.

Notice that the packaging element in the pom.xml file must be set to jar to build a JAR file:

|  |  |
| --- | --- |
| 1 | <packaging>jar</packaging> |

If we don't include this element, it also defaults to jar.

In case we want to build a WAR file, change the packaging element to war:

|  |  |
| --- | --- |
| 1 | <packaging>war</packaging> |

And leave the container dependency off the packaged file:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-tomcat</artifactId>      <scope>provided</scope>  </dependency> |

After executing the Maven package phase, we'll have a deployable WAR file.

### ****What Are Possible Sources of External Configuration?****

Spring Boot provides support for external configuration, allowing us to run the same application in various environments. We can use properties files, YAML files, environment variables, system properties, and command-line option arguments to specify configuration properties.

We can then gain access to those properties using the @Value annotation, a bound object via [the @ConfigurationProperties annotation](https://www.baeldung.com/configuration-properties-in-spring-boot), or the Environment abstraction.

Here are the most common sources of external configuration:

* Command-line properties: Command-line option arguments are program arguments starting with a double hyphen, such as –server.port=8080. Spring Boot converts all the arguments to properties and adds them to the set of environment properties.
* Application properties: Application properties are those loaded from the application.properties file or its YAML counterpart. By default, Spring Boot searches for this file in the current directory, classpath root, or their config subdirectory.
* Profile-specific properties: Profile-specific properties are loaded from the application-{profile}.properties file or its YAML counterpart. The {profile} placeholder refers to an active profile. These files are in the same locations as, and take precedence over, non-specific property files.

### What does it mean that Spring Boot supports relaxed binding?

Relaxed binding in Spring Boot is applicable to [the type-safe binding of configuration properties](https://www.baeldung.com/configuration-properties-in-spring-boot).

With relaxed binding, the key of an environment property doesn't need to be an exact match of a property name. Such an environment property can be written in camelCase, kebab-case, snake\_case, or in uppercase with words separated by underscores.

For example, if a property in a bean class with the @ConfigurationProperties annotation is named myProp, it can be bound to any of these environment properties: myProp, my-prop, my\_prop, or MY\_PROP.

### ****What Is Spring Boot Actuator Used For?****

Essentially, Actuator brings Spring Boot applications to life by enabling production-ready features. These features allow us to monitor and manage applications when they're running in production.

Integrating Spring Boot Actuator into a project is very simple. All we need to do is to include the spring-boot-starter-actuator starter in the pom.xml file:

|  |  |
| --- | --- |
| 1  2  3  4 | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-actuator</artifactId>  </dependency> |

Spring Boot Actuator can expose operational information using either HTTP or JMX endpoints. Most applications go for HTTP, though, where the identity of an endpoint and the /actuator prefix form a URL path.

Here are some of the most common built-in endpoints Actuator provides:

* auditevents: Exposes audit events information
* env: Exposes environment properties
* health: Shows application health information
* httptrace: Displays HTTP trace information
* info: Displays arbitrary application information
* metrics: Shows metrics information
* loggers: Shows and modifies the configuration of loggers in the application
* mappings: Displays a list of all @RequestMapping paths
* scheduledtasks: Displays the scheduled tasks in your application
* threaddump: Performs a thread dump

## **Citez quelques avantages du framework Spring**

Avant que Spring ne prenne de l’importance, JEE était le choix évident pour développer une application. Mais ce dernier présentait de nombreuses limitations. Le code n’était pas assez modulaire, et plus l’application devenait importante, plus il devenait difficile d’identifier ses composants séparément.  De plus, c’était un framework très lourd, qui avait tendance a dégrader les performances.

Spring a réussi a se défaires de tous les écueils du framework J2EE en un claquement de doigt. Spring est uniquement composé de « plain old Java objects » (c’est-à-dire de bons vieux objets Java, bien plus faciles à utiliser).

Enfin, le framework Spring peut-être utilisé pour le développement d’applications à interface graphique, d’applications web, d’applets, etc.

Bonnes réponses :

Cite l’avantage majeur Spring, à savoir la création d’une application composée de « plain old Java objects »

Explique les désavantages des méthodes utilisées auparavant

Réponses alarmantes :

Ne connaît pas les avantages du framework

N’a qu’une idée très basique des composants de Spring

## **Que pouvez-vous dire à propos de l’acronyme S.O.L.I.D, bien souvent utilisé en programmation orienté objet ?**

Michael Feathers est connu pour avoir popularisé l’acronyme S.O.L.I.D, bien que Robert Martin soit l’inventeur de cette ligne directrice créée pour désigner les 5 principes de la programmation orientée objet. S.O.L.I.D signifie :

·        « Single responsibility principle » (Responsabilité unique)

·        « Open/Closed principle » (Ouvert/Fermé)

·        « Liskov substitution principle » (Substitution de Liskov)

·        « Interface segregation principle » (Ségrégation des interfaces)

·        « Dependency inversion principle » (Inversion des dépendances)

Le principe de responsabilité unique préconise qu’une classe ne doit changer que pour une unique raison, et que chaque classe doit avoir une unique responsabilité. Cela signifie que l’on peut mettre un tas de choses dans une seule classe, mais il ne s’agit certainement pas d’une bonne pratique.

Le principe ouvert/fermé prévoit que toute classe doit être ouverte à l’extension mais fermée à la modification. Il faut uniquement créer des getters et des setters là où ils sont nécessaires, et garder privées les variables.

Selon la substitution de Liskov, les objets d’un programme doivent être remplaçables par leurs sous-types, sans entraîner une erreur.

La ségrégation des interfaces préconise qu’aucun client ne doit dépendre de méthodes qu’il n’utilise pas.

L’inversion des dépendances (à ne pas confondre avec l’injection des dépendances) prévoit que les modules de haut niveau ne doivent pas dépendre des modules de bas niveau, et que les abstractions ne doivent pas dépendre des détails (mais que les détails doivent dépendre des abstractions).

Bonnes réponses :

Connaître les cinq principes

Avoir une idée claire de ces principes, étayer avec des exemples

Prouver une bonne compréhension des enjeux de la programmation orientée objet

Réponses alarmantes :

N’avoir jamais entendu parlé de l’acronyme S.O.L.I.D.

Incapacité à donner des exemples concrets

## **Qu’est-ce qu’un conteneur IoC permet de faire sur Spring ?**

IoC signifie « inversion of control » (inversion de contrôle).

L’IoC permet de faire en sorte que le framework prenne en charge l’exécution principale du programme.

Dans le cas de Spring, le conteneur IoC est implémenté au niveau de l’ApplicationContext Interface. Il prend tout en charge : initialisation, configuration et assemblage des beans tout au long de leur cycle de vie.

La configuration de l’IoC peut se faire par le biais de fichiers de configuration XML, d’annotations Java ou directement dans le code source Java.

Bonne réponses :

Démontre une bonne connaissance de l’IoC

Expliquer l’implémentation de l’IoC par Spring grâce à l’injection des dépendances

Réponses alarmantes :

N’a jamais entendu parlé de l’IoC

A du mal à comprendre les enjeux de l’IoC et n’évoque pas du tout la question de l’injection des dépendances

## **Quelles sont les différences entre l’inversion de contrôle et une implémentation de classes POJO (plain old Java objects) ?**

Imaginez que vous avez une interface, et une classe qui implémente cette interface. Cette classe aurait également des méthodes qui écrasent les implémentations de l’interface. Pour exécuter une fonction en particulier, il faudra créer un objet de cette classe, puis exécuter la fonction depuis l’instance de cet objet. Voilà le scénario classique en utilisant des classes POJO.

A l’inverse, si on vient à utiliser le conteneur IoC du Spring, la création d’un objet devient déléguée à une entité externe (Spring en l’occurrence). Un objet nous est alors fourni par le conteneur IoC, et on peut directement appelé les méthodes à l’intérieur.

Bonne réponses :

Explique que la meilleure approche serait de poursuivre l’implémentation

Mauvaises réponses :

Ne parvient pas à différencier ces deux types d’implémentation

Incapacité à expliquer le rôle du conteneur IoC

## **Pourquoi la configuration en utilisant les annotations est-elle préférable à la configuration XML ?**

Depuis la version 2.5 de Spring, l’injection des dépendances peut être réalisable en utilisant les annotations. Cela constitue une avancée majeure car un grand nombre de personnes détestaient l’ancienne norme, à savoir la configuration par le biais de fichiers XML. Grâce à la configuration basée sur les annotations, les dépendances peuvent être injectées directement dans la classe du composant en annotant une classe, un champ ou une méthode pertinente.

Bonne réponses :

Capacité à différencier les deux méthodes

Réponses alarmantes :

Mauvaise connaissance du sujet, difficulté à citer différentes annotations de configuration

## **Quels sont les atouts et les limites de la programmation réactive ?**

La programmation réactive permet de gagner du temps à l’exécution, de simplifier l’écriture de certaines applications, d’éviter certains bugs (oubli de mise à jour de valeur par exemple) et de ne plus avoir à tenir compte de l’ordre de déclenchement des événements.

Malgré ces nombreux avantages, la programmation réactive présente bien évidemment quelques limites. Parmi elles, la difficulté de son implémentation, ou sa nature asynchrone (rendant plus difficile la compréhension et le suivi d’un flux).

Bonnes réponses :

Présente un bon recul par rapport aux avantages et aux inconvénients de la programmation réactive

Réponses alarmantes :

Ne connaît pas la notion de programmation réactive

N’évoque pas le caractère asynchrone de la programmation réactive