Développement Logiciel Java

03/02/2025

Les blocs d’initialisation

Le bloc static: Il s’agit d’un mot clé *static* et d’accolades, à l’intérieur de la classe.

Ce bloc s’exécute une seule fois par exécution du programme, lors du chargement de la **classe** en mémoire. Il ne peut faire référence qu’aux variables *static* et ne peut pas utiliser *this*.

Le bloc d’initialisation d’instance: Il s’agit simplement d’un jeu d’accolades à l’intérieur de la classe (de préférence tout au début). Ce bloc s’exécute une seule fois par initialisation d’objet, avant les constructeurs mais après l’appel aux constructeurs parents. Il peut utiliser les paramètres d’instances et le mot-clé *this*.

Le bloc d’initialisation peut servir à factoriser un comportement qui doit s’effectuer dans tous les constructeurs.

Le pattern Singleton

A l'aide des modificateurs de visibilité et des méthodes statiques, il est possible d'implémenter une classe suivant le design pattern *singleton*.

Il faut:

- Rendre le constructeur privé pour en empêcher l'instanciation.
- Garder une référence sur l'instance unique existante dans une variable statique privée.
- Créer une méthode statique publique chargée de créer et / ou de renvoyer l'instance unique.

L'instance Singleton est utilisée pour certains objets techniques qui ne peuvent exister qu'une seule fois en mémoire, sous peine de se faire concurrence sur l'accès aux ressources par exemple.

Le pattern Builder

Quand une classe contient beaucoup de paramètres, il peut être utile de recourir au design pattern *builder*.

Il s’agit d’une seconde classe dont le but est de construire facilement la première.

La méthode finalize

- La méthode finalize est exécutée juste avant la désallocation de l'objet et doit contenir les opérations à effectuer urgemment pour éviter les fuites de mémoire (par exemple, fermer un flux sur un fichier ou une connexion à une BDD).
- La méthode finalize est un dernier recours car elle impacte les performances. Il faut lui préférer une méthode *close* que le consommateur de notre classe est responsable d'appeler lorsqu'il a fini son traitement.

Le mot-clé *this*

- Référence l'instance en cours depuis l'intérieur de la classe.
- Accessible uniquement depuis les constructeurs, le bloc d'initialisation d'instance, et les méthodes non statiques.
- S'utilise avec un point pour référencer les méthodes et les paramètres de la classe (*this.property*)
- En général facultatif, mais il permet parfois de lever des ambiguité si d'autres variables portent le même nom qu'un paramètre.
- Utilisé avec des parenthèses, il correspond à un constructeur de la classe.

Les méthodes

```
[Attribut visibilité][Modificateur]<type de retour><Nom>([type

param], [type param2],...)<throws exception1, exception2>{

   <Code>;

   <Code>;

   //...

}
```

```
public static Double getValue(Double x, Double y) throws IOException {

      <code>;

}
```

Les méthodes - Modificateur

Le modificateur peut être:

- Static, pour associer la méthode à la classe et non aux instances;
- final
- abstract

Ces deux derniers modificateurs sont liés au mécanismes d'héritage et seront détaillés plus tard.

Les méthodes - Types de retour

Le type de retour indique ce que la méthode renverra à la fin de son traitement.

Si un type de retour (autre que *void*) est déclaré, la méthode DOIT absolument renvoyer une valeur en accord, dans tous les cas (dans toutes ses branches).

Le retour peut être:

- void : la méthode ne renvoie rien
- Un type référence
- Un type primitif.

Le nom de la méthode

La convention est de nommer les méthodes en camelCase. Comme pour le nom des attributs, les noms de méthodes doivent:

- Commencer par une lettre ou un _
- Contenir des lettres, des chiffres ou des _

Les noms de méthodes devraient:

- Ne pas contenir de caractères accentués
- Avoir un nom suffisamment descriptif même s'il est long.

Les arguments

- Les passages de paramètres se font par “recopie” du contenu de la variable.
- Pour un type primitif, c’est donc la valeur qui est recopiée dans le paramètre, et passée à la méthode. Changer cette valeur ne changera pas la valeur de la variable originale. Si l’on veut renvoyer un changement, il convient d’utiliser une valeur de retour.
- Pour un type référence c’est l’adresse de l’objet qui est recopiée et passée en méthode. C’est donc le même objet qui est manipulé dans la méthode et dans le code appelant. Changer son état changera l’état de l’objet partout.
- Les arguments de méthodes sont des variables locales à la méthode. Ils n’existent qu’entre les accolades de la méthode.

Les Exceptions

Les système d'exception sert à remonter des cas problématiques tels que:

- Erreurs de développement (tenter d'écrire en dehors de la place disponible dans un tableau)
- Erreurs d'entrée des données utilisateur (recevoir des lettres alors que l'on attend des chiffres)
- Erreurs liés au système (manque de place sur les disques, interruption réseau...)

Les deux premiers cas devraient pouvoir être évités par les alertes de l'IDE, les tests unitaires, et la validation des données. Le troisième est imprévisible par nature.

Les Exceptions

Lorsqu'un dysfonctionnement est détecté:

La méthode émettrice d'une Exception alloue un objet de type (ou héritant du type) *Throwable*, contenant des messages et informations précises sur le problème rencontré.

L'erreur est levée avec le mot-clé *throw*.

La méthode est interrompue immédiatement.

La JVM remonte la pile des appels de méthodes à la recherche d'un endroit capable de gérer cette exception (*catch*).

Si aucune gestion de cette exception n'existe, la JVM interrompt le programme en cours avec un message d'erreur au système d'exploitation.

Les Exceptions

Lorsque l'on consomme une méthode connue pour lever des exceptions, on peut:

- Soit les laisser passer, en le déclarant à notre tour dans la signature de notre méthode
- Soit les traiter au moyen d'un bloc *try { } catch {}*;

```
try {
     doSomethingRisky();
} catch (Exception e) {
     Logger.error("An unexpected error occured, please try again later");
}
```

Les Exceptions

Le bloc *finally* vient en complément des blocs *try* et *catch*. Il permet d'exécuter des instructions après les deux autres blocs, qu'il y ait eu ou non une exception levée.

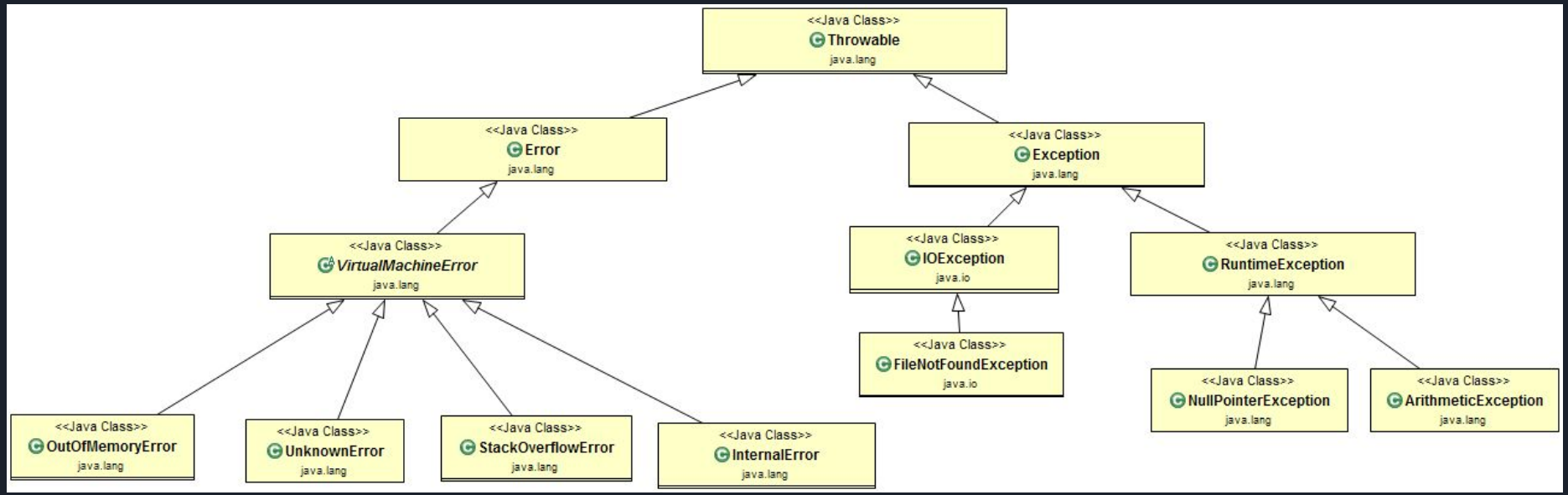
Il sert à nettoyer des objets qui ont été créés dans le blocs et qui risqueraient de rester en mémoire en cas d'erreur.

Les Exceptions

Il est possible de chaîner les blocs *catch* pour traiter différents types d'exceptions.

Il est possible (mais pas recommandé) d'utiliser l'héritage, pour traiter de manière globale toute une famille d'exceptions.

Les Exceptions



https://cdn.rollbar.com/wp-content/uploads/2021/07/java-exceptions-hierarchy-example.png

Les Exceptions

Certains cas d'erreurs sont imprévisibles par nature et java considère qu'ils ne doivent pas être considérés / traités préventivement. Ce sont les *unchecked* exceptions:

- Toutes les sous-classes de la classe *Error*
- Les sous-classes de la classe *RuntimeException*.

Les autres, sont les *checked* exceptions. Lorsqu'une méthode déclare une de ces exceptions, le code appelland DOIT soit la traiter, soit la déclarer à son tour.

Les Exceptions

Si le cas d'erreur traité par votre programme ne correspond pas à une exception existante, il est possible et conseillé de créer votre propre exception, héritant de l'Exception java. Cela permet de décrire précisément le problème, d'y ajouter des paramètres supplémentaires pour aider à la résolution par exemple.

TP

Nous allons améliorer notre application de gestion de TODO List.

- Pour accéder aux données (Users, Tasks), nous allons créer une classe dédiée DatabaseAccess.
  - Cette classe doit implémenter le design-pattern Singleton
  - Pour simuler une base de données, elle doit s'initialiser avec une liste de Tasks et une liste d'Users.
  - Elle doit contenir les méthodes permettant de trouver, supprimer, et ajouter des Tasks et des Users.
  - Créer une exception dédiée pour le cas où l'on cherche à obtenir / modifier un élément qui n'existe pas.
- Chaque modele (Tasks, Users) doit contenir un compteur d'instance. Utiliser les blocs statiques pour instancier ce compteur, les blocs d'instances pour l'incrémenter, et utiliser ce compteur pour générer l'ID.
- Créer un *TaskBuilder* pour aider à la création de tâches.