Exceptions en Java

Qu'est-ce qu'une exception?

Une exception est un événement indésirable ou inattendu, qui se produit pendant l'exécution d'un programme, c'est-à-dire au moment de l'exécution, qui perturbe le flux normal des instructions du programme.

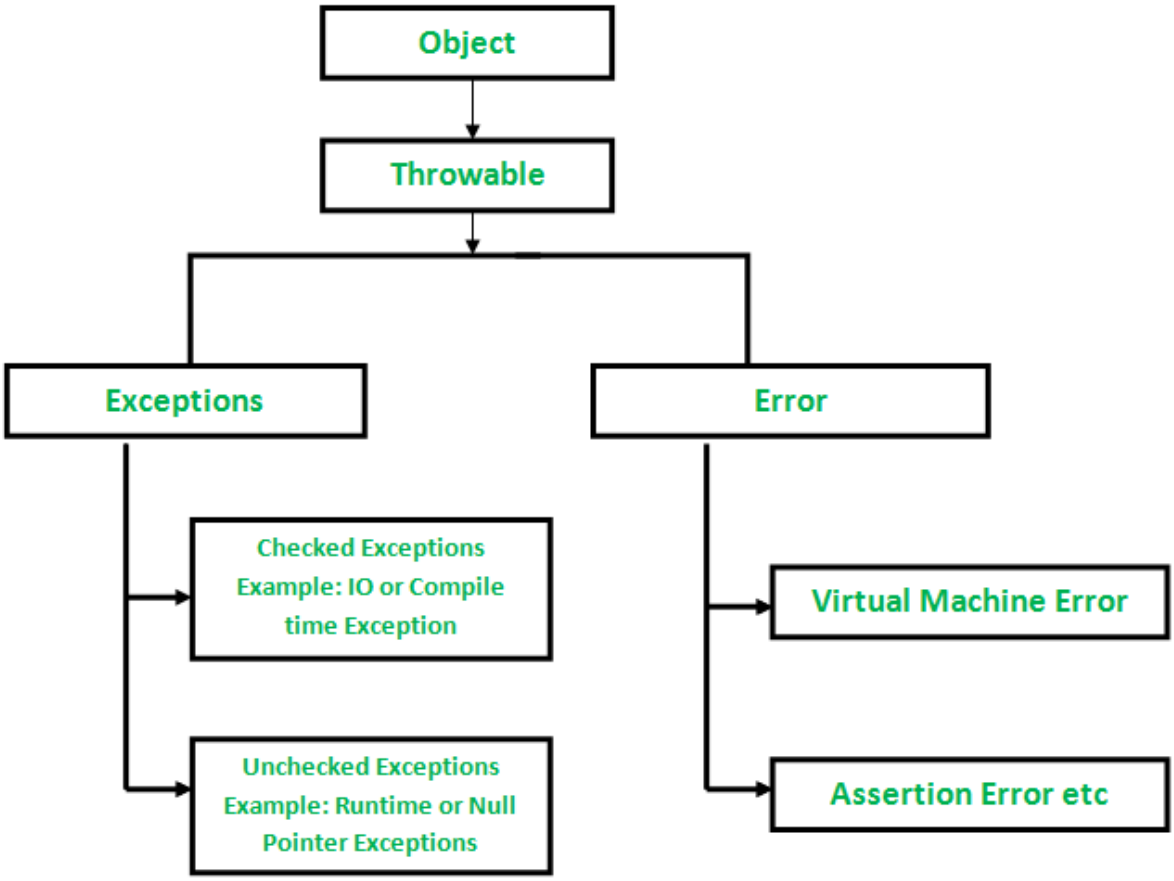
Erreur vs exception

Erreur: une erreur indique un problème grave qu'une application raisonnable ne devrait pas essayer d'attraper.

Exception: une exception indique des conditions qu'une application raisonnable pourrait essayer de détecter.

Hiérarchie des exceptions

Tous les types d'exceptions et d'erreurs sont des sous-classes de la classe **Throwable** , qui est la classe de base de la hiérarchie. Une branche est dirigée par **Exception** . Cette classe est utilisée pour des conditions exceptionnelles que les programmes utilisateur doivent intercepter. NullPointerException est un exemple d'une telle exception. Une autre branche, **Error,** est utilisée par le système d'exécution Java (JVM) pour indiquer les erreurs liées à l'environnement d'exécution lui-même (JRE). StackOverflowError est un exemple d'une telle erreur.



Pour les exceptions vérifiées et non contrôlées, voir Exceptions vérifiées et non contrôlées

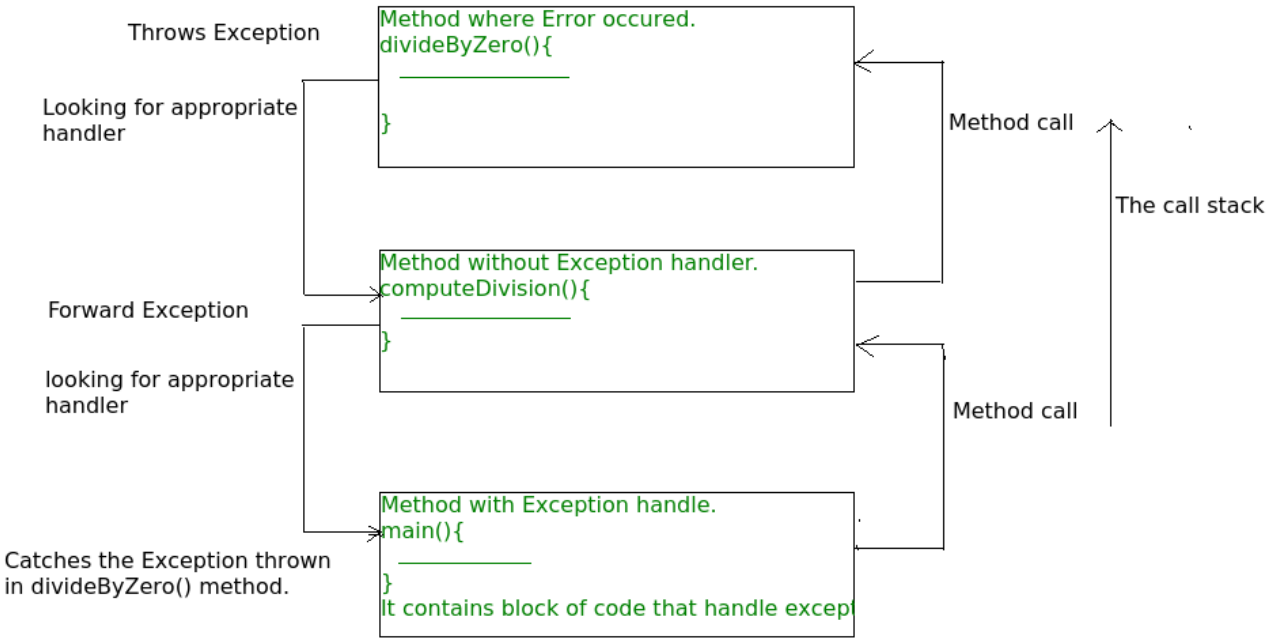
Comment JVM gère une exception?

Gestion des exceptions par défaut: chaque fois que dans une méthode, si une exception s'est produite, la méthode crée un objet appelé objet d'exception et le transmet au système d'exécution (JVM). L'objet exception contient le nom et la description de l'exception, ainsi que l'état actuel du programme où l'exception s'est produite. La création de l'objet exception et sa gestion vers le système d'exécution sont appelées lancer une exception. Il peut y avoir la liste des méthodes qui ont été appelées pour arriver à la méthode où l'exception s'est produite. Cette liste ordonnée des méthodes s'appelle la **pile d'appels.** Maintenant, la procédure suivante se produira.

- Le système d'exécution recherche dans la pile d'appels pour trouver la méthode qui contient le bloc de code qui peut gérer l'exception survenue. Le bloc du code est appelé **gestionnaire d'exceptions** .
- Le système d'exécution démarre la recherche à partir de la méthode dans laquelle l'exception s'est produite, passe par la pile d'appels dans l'ordre inverse dans lequel les méthodes ont été appelées.
- S'il trouve le gestionnaire approprié, il lui transmet l'exception qui s'est produite. Un gestionnaire approprié signifie que le type de l'objet exception levé correspond au type de l'objet exception qu'il peut gérer.
- Si le système d'exécution recherche toutes les méthodes de la pile d'appels et n'a pas pu trouver le gestionnaire approprié, le système d'exécution transfère l'objet d' **exception au gestionnaire d'exceptions par défaut** , qui fait partie du système d'exécution. Ce gestionnaire imprime les informations d'exception au format suivant et termine le programme de manière **anormale** .

- `Exception dans le thread "xxx" Nom de l'exception: Description`
- `... ..... .. // Pile d'appels`

Consultez le diagramme ci-dessous pour comprendre le flux de la pile d'appels.



The call stack and searching the call stack for exception handler.

Exemple :

```
// Java program to demonstrate how exception is thrown.

class ThrowsExecp{


    public static void main(String args[]){


        String str = null;

        System.out.println(str.length());


    }

}
```

Production :

```
Exception dans le thread "main" java.lang.NullPointerException
    sur ThrowsExecp.main (File.java:8)
```

Voyons un exemple qui illustre comment le système d'exécution recherche le code de gestion des exceptions approprié sur la pile d'appels:

```
// Java program to demonstrate exception is thrown
// how the runTime system searches th call stack
// to find appropriate exception handler.
class ExceptionThrown
{
    // It throws the Exception(ArithmeticException).
    // Appropriate Exception handler is not found within this method.
    static int divideByZero(int a, int b){

        // this statement will cause ArithmeticException(/ by zero)
        int i = a/b;

        return i;
    }
   static int computeDivision(int a, int b) {

        int res =0;

        try
        {
```

```
          res = divideByZero(a,b);
        }
        // doesn't matches with ArithmeticException
        catch(NumberFormatException ex)
        {
           System.out.println("NumberFormatException is occured");
        }
        return res;
    }

    // In this method found appropriate Exception handler.
    // i.e. matching catch block.
    public static void main(String args[]){
        int a = 1;
        int b = 0;
     try
        {
            int i = computeDivision(a,b);
                  }
      catch(ArithmeticException ex)
        {
            // getMessage will print description of exception(here / by zero)
            System.out.println(ex.getMessage());
        }    } }
```

Production :

```
/ par zéro.
```

Comment Programmer gère une exception?

Gestion des exceptions personnalisée: La gestion des exceptions Java est gérée via cinq mots clés: **essayer** , **intercepter** , **lancer** , **lancer** et **finally** . En bref, voici comment ils fonctionnent. Les instructions de programme qui, selon vous, peuvent déclencher des exceptions sont contenues dans un bloc try. Si une exception se produit dans le bloc try, elle est levée. Votre code peut intercepter cette exception (en utilisant le bloc catch) et la gérer de manière rationnelle. Les exceptions générées par le système sont automatiquement levées par le système d'exécution Java. Pour lever manuellement une exception, utilisez le mot clé throw . Toute exception levée d'une méthode doit être spécifiée comme telle par un lancerclause. Tout code qui doit absolument être exécuté après la fin d'un bloc try est placé dans un bloc finally.

Article détaillé: Contrôler le flux dans le bloc try catch finally

Besoin d'une clause try-catch (gestion des exceptions personnalisée)

Considérez le programme java suivant.

```
class GFG {

    public static void main (String[] args) {

        // array of size 4.

        int[] arr = new int[4];


        // this statement causes an exception

        int i = arr[4];

        System.out.println("Hi, I want to execute");

    }

}
```

Production :

```
Exception dans le thread "principal"
java.lang.ArrayIndexOutOfBoundsException: 4

    sur GFG.main (GFG.java:9)
```

Explication: Dans l'exemple ci-dessus, un tableau est défini avec une taille, c'est-à-dire que vous ne pouvez accéder aux éléments que de l'index 0 à 3. Mais vous essayez d'accéder aux éléments de l'index 4 (par erreur), c'est pourquoi il lève une exception. Dans ce cas, JVM termine **anormalement** le programme . L'instruction *System.out.println («Salut, je veux exécuter»);* ne s'exécutera jamais. Pour l'exécuter, nous devons gérer l'exception à l'aide de try-catch. Par conséquent, pour continuer le déroulement normal du programme, nous avons besoin d'une clause try-catch.

Comment utiliser la clause try-catch

```
essayez {

// bloc de code pour surveiller les erreurs

// le code que vous pensez peut déclencher une exception

}

catch (ExceptionType1 exOb) {

// gestionnaire d'exceptions pour ExceptionType1

}

catch (ExceptionType2 exOb) {

// gestionnaire d'exceptions pour ExceptionType2

}

// optionnel

finally {

// bloc de code à exécuter après la fin du bloc try

}
```

Points à retenir :

- Dans une méthode, il peut y avoir plus d'une instruction qui peut lever une exception, alors placez toutes ces instructions dans son propre bloc **try** et fournissez un gestionnaire d'exceptions séparé dans son propre bloc **catch** pour chacune d'entre elles.
- Si une exception se produit dans le bloc **try** , cette exception est gérée par le gestionnaire d'exceptions qui lui est associé. Pour associer le gestionnaire d'exceptions, nous devons mettre le bloc **catch** après lui. Il peut y avoir plusieurs gestionnaires d'exceptions. Chaque bloc **catch** est un gestionnaire d'exceptions qui gère l'exception du type indiqué par son argument. L'argument, ExceptionType, déclare le type de l'exception qu'il peut gérer et doit être le nom de la classe qui hérite de la classe **Throwable** .
- Pour chaque bloc try, il peut y avoir zéro ou plusieurs blocs catch, mais **un seul** bloc finalement.
- Le bloc finally est facultatif, il est toujours exécuté, qu'une exception se soit produite ou non dans le bloc try. Si une exception se produit, elle sera exécutée après **les blocs try et catch.** Et si aucune exception ne se produit, elle sera exécutée après le bloc **try** . Le bloc finally en java est utilisé pour mettre des codes importants tels que le code de nettoyage, par exemple la fermeture du fichier ou la fermeture de la connexion.