Programmation Orienté Objet

Chapitre 5 : Les exceptions

Omar El Midaoui

Décembre 2021

Génération et gestion d'exception

1. Les exceptions servent à gérer les erreurs qui peuvent survenir durant l'exécution d'un programme
2. Les exceptions sont utilisés pour repérer les parties "dangereuses / à risque" dans un programme.
3. Lorsqu'une exception survient :
   - un objet représentant cette exception est créé;
   - cet objet est **jeté (thrown)** dans la méthode ayant provoqué l'erreur.
4. Cette méthode peut choisir :
   - de gérer l'exception elle-même,
   - de la passer sans la gérer.

De toutes façons, l'exception est captée (caught) et traitée en dernier recours par l'environnement d'exécution java.

Génération et gestion d'exception (2)

1. Les exceptions peuvent être générées :
   - par l'environnement d'exécution java
   - manuellement par du code
2. Les exceptions jetées (ou levées) par l'environnement d'exécution résultent de violations des règles du langage ou des contraintes de cet environnement d'exécution.

Structure générale du traitement des exceptions

```
try {
//bloc de code a surveiller
//peut lever une ou plusieurs exceptions
}
catch (ExceptionType1 exceptObj) {
//Traitement de l'exception du type1
}
catch (ExceptionType2 exceptObj) {
//Traitement de l'exception du type2
}
finally {
//code a exécuter avant de sortir (avec ou sans le traitement des
exceptions)
}
```

Exception non gerée

1. Considérons le code suivant ou une division par zéro n'est pas gérée par la programme :

ExcepDiv0.java

```
class ExcepDiv0 {
public static void main(String args[]) {
int d = 0;
int a = 42 / d; } }
```

2. Lorsque l'environnement d'exécution essaie d'exécuter la division, il construit un nouvel objet exception afin d'arrêter le code et de gérer cette condition d'erreur.
3. L'environnement d'exécution affiche la valeur en String de l'exception et la trace de la pile d'appels :

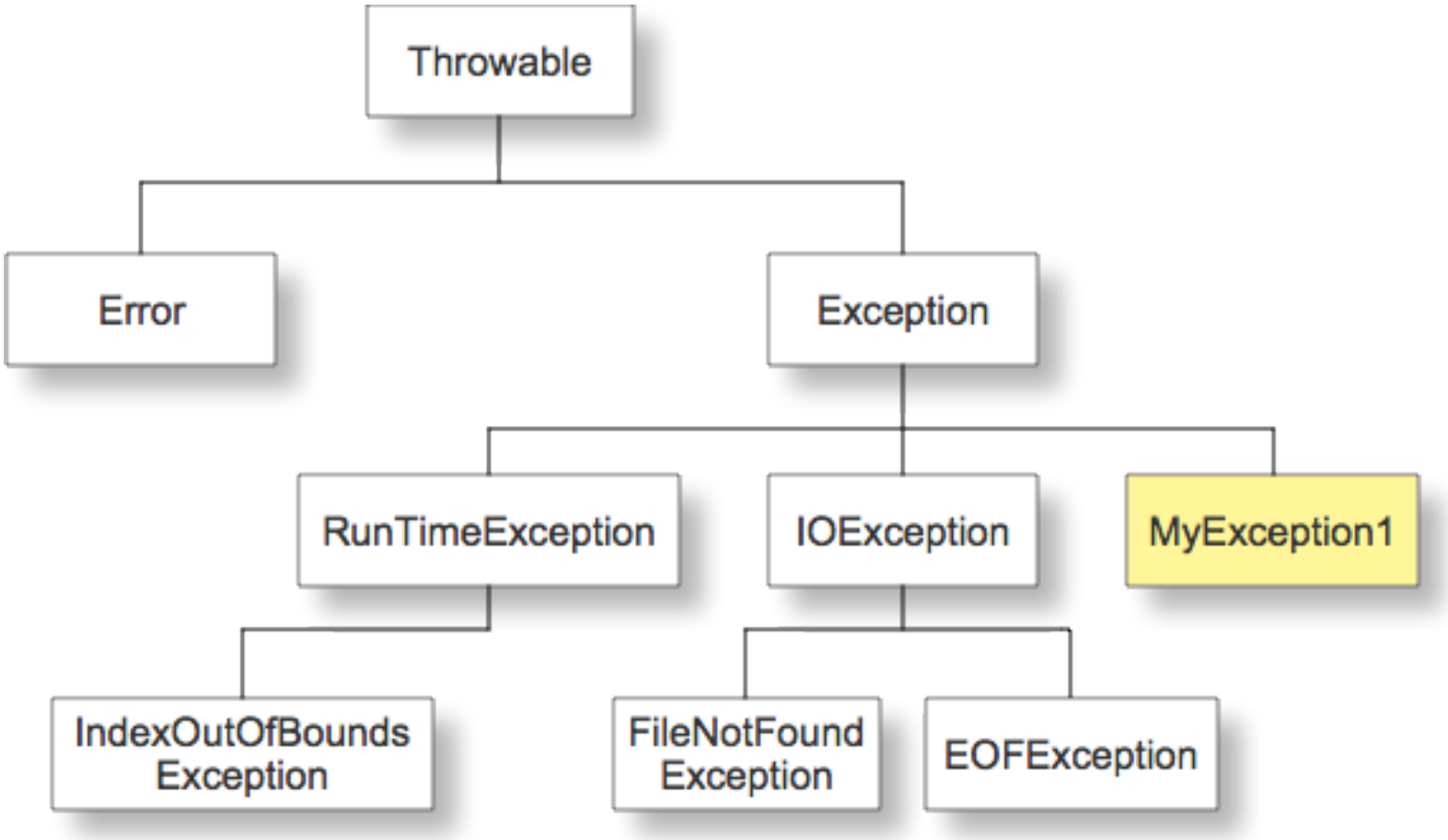
```
> java ExcepDiv0
java.lang.ArithmeticException: / by zero
at ExcepDiv0.main(ExcepDiv0.java:4)
```

Types d'exceptions

1. Une classe est au sommet de la hiérarchie des exceptions : **Throwable**
2. Deux sous-classes de Throwable :
   - Exception : conditions exceptionnelles que les programmes utilisateur devraient traiter.
   - Error : exceptions catastrophiques que normalement seul l'environnement d'exécution devrait gérer.
3. Une sous-classe d'Exception, RuntimeException, pour les exceptions de l'environnement d'exécution.

Types d'exceptions (2)

- En java les exceptions sont des objets
- toute exception doit être une instance d'une sous-classe de la classe java.lang.Throwable



Types d'exceptions (3)

```
Exception
    ClassNotFoundException
    CloneNotSupportedException
    IllegalAccessException
    InstantiationException
    InterruptedException
    NoSuchFieldException
    NoSuchMethodException
    RuntimeException
        ArithmeticException
        ArrayStoreException
        ClassCastException
        IllegalArgumentException
            IllegalThreadStateException
            NumberFormatException
        IllegalMonitorStateException
        IllegalStateException
        IndexOutOfBoundsException
            ArrayIndexOutOfBoundsException
            StringIndexOutOfBoundsException
```

Instructions try et catch

1. Un bloc try est destiné a être protégé, gardé contre toute exception susceptible de survenir.
2. **try**{ . . . } délimite un ensemble d'instructions susceptibles de déclencher une(des) exception(s) pour la(les)quelles une gestion est mise en oeuvre
3. Juste après un bloc try, il faut mettre un bloc **catch** qui sert de gestionnaire d'exception. Le paramètre de l'instruction catch indique le type et le nom de l'instance de l'exception : **catch(TypeDexception e)** { . . . }
4. Cela permet d'intercepter ("attraper") les exceptions dont le type est spécifié et d'exécuter alors du code spécifique

Instructions try et catch (2)

ExcepDiv0.java

```
class ExcepDiv0 {
public static void main(String args[]) {
try {
int d = 0;
int a = 42 / d; }
catch (ArithmeticException e) {
System.out.println("Div par zero"); }
}
}
```

Instructions catch multiples

1. On peut gérer plusieurs exceptions a la suite l'une de l'autre.
2. Lorsqu'une exception survient, l'environnement d'exécution inspecte les instructions catch les unes après les autres, dans l'ordre ou elles ont été écrites.
3. Il faut donc mettre les exceptions les plus spécifiques d'abord.

Instructions catch multiples (2)

Excepmultiple.java

```
class Excepmultiple {
public static void main(String args[]) {
try {
String s = args[0];
int a = Integer.parsInt(" 123 ");
int b = Integer.parsInt( s );
System.out.println( a/b );
}
 catch (ArrayIndexOutOfBoundsException ex0)  {
//Traitement de l'exception en cas de dépassement de tableau
}
```

Instructions catch multiples (3)

Excepmultiple.java

```
catch (NumberFormatException ex1) {
//Traitement de l'exception en cas d'erreur de parsing
}
catch (Exception ex2) {
//Traitement des exceptions générales (telle que la division par 0)
}
}
}
```

Instructions catch multiples (4)

1. S'il n'y a pas de block catch alors le block finally est requis, sinon il est optionnel.
2. On peut attraper plusieurs exceptions dans un même block try, chaque classe d'exception a son propre traitement.
3. Les instructions d'un bloc try situées après une levée d'exception ne sont pas exécutées.
4. L'ordre des blocks catch est très important : Les classes les plus générales d'exception sont placées en dernier.

Instruction finally

- Les clauses catch sont suivies de manière optionnelle par un bloc finally qui contient du code qui sera exécuté quelle que soit la manière dont le bloc try a été quitté
- Le bloc finally permet de spécifier du code dont l'exécution est garantie quoi qu'il arrive :
  - le bloc try s'exécute normalement sans qu'aucune exception ne soit levée
  - le bloc try lève une exception attrapée par l'un des blocs catch.
  - le bloc try lève une exception qui n'est attrapée par aucun des blocs catch qui le suivent.

Instruction finally (2)

- Intérêt double :
  - permet de rassembler dans un seul bloc un ensemble d'instructions qui autrement auraient du être dupliquées
  - permet d'effectuer des traitements après le bloc try, même si une exception a été levée et non attrapée par les blocs catch

```
...
try {
// ouvrir un fichier
// effectuer des traitements
// susceptibles de lever une exception
// fermer le fichier
}
catch (CertaineException ex1){
// traiter l'exception
// fermer le fichier
}
catch (AutreTypeException ex2){
// traiter l'exception
// fermer le fichier
}
```

```
...
try {
// ouvrir un fichier
// effectuer des traitements
// susceptibles de lever une exception
}
catch (CertaineException ex1){
// traiter l'exception
}
catch (AutreTypeException ex2){
// traiter l'exception
}
finally {
// fermer le fichier
}
```

Instruction throw

1. Elle permet de générer une exception, via un appel de la forme :
   throw TrhowableInstance ;
2. Cette instance peut être créé par un **new** ou être une instance d'une exception déjà existante (sous-classe de Throwable).
3. Le flux d'exécution est alors stoppé et le bloc try immédiatement englobant est inspecté, afin de voir s'il possède une instruction catch correspondante à l'instance générée.
4. Si ce n'est pas le cas, le $2^{\text{ième}}$ bloc try englobant est inspecté ; et ainsi de suite.

Instruction throw (2)

ThrowDemo.java

```
class Article { double prixVente;
        double vendre(int qte) {
        if(stock < qte){throw new IllegalArgumentException("Stock insuffisant"); }
        else            {
                            stock-=qte;
                            return qte * prixVente;}
                                    }
        public static void main(String args[]) {
                    try { a1.vendre(20);}
                    catch(IllegalArgumentException e1) {
                            e1.printStackTrace();
        // ou bien :        System.out.println(e1.getMessage());
                                                } } }
```

Instruction throws (3)

1. Si une méthode est susceptible de générer une exception qu'elle ne gère pas, elle doit le spécifier, de façon que ceux qui l'appellent puissent se prémunir contre l'exception.
2. L'instruction throws est utilisée pour spécifier la liste des exceptions qu'une méthode est susceptible de générer.
3. Pour la plupart des sous-classes d'Exception, le compilateur forcera à déclarer quels types d'exception peuvent être générées (sinon, le programme ne compile pas).
4. Cette règle ne s'applique pas à Error, RuntimeException ou à leurs sous-classes.

Instruction throws (2)

L'exemple suivant ne se compilera pas :

ThrowsDemo1.java

```
class ThrowsDemo1 {
static void proc() {
System.out.println("dans proc()");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
proc();
}
}
```

Ce programme ne se compilera pas parce que :

- proc() doit déclarer qu'elle peut générer IllegalAccessException
- main() doit avoir un bloc try/catch pour gérer l'exception en question.

Instruction throws (3)

L'exemple correct est :

ThrowsDemo1.java

```
class ThrowsDemo1 {
static void proc() throws IllegalAccessException {
System.out.println("dans proc()");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
proc();
 }
catch(IllegalAccessException e) {
System.out.println(e + " attrapée");
 }
} }
```

Définition d'une classe d'exception

Définition d'une classe d'exception

On crée une exception comme n'importe quelle autre classe :

ExcptDiv0.java

```
class ExcptDiv0 extends Exception {
public ExcptDiv0 (String s) {
super(s);
}
}
...
public void divise(double x, double y) throws ExceptDiv0 {
if (y = 0)
    throw new ExceptDiv0("Atention !! division par zero ");
return x/y;
}
```

Le programme qui utilisera cette méthode doit gérer l'exception et alors mettre la méthode divise dans un bloc try.