Compléments de POO en Java

L2-MIM4A1

Année 2019/2020

CM 3

Yann Mathet yann.mathet@unicaen.fr

Les « exceptions »

- Mécanisme permettant un traitement amélioré des « erreurs »
- Une exception est un événement inattendu, non souhaité.
- Exemples :
  - Division par zéro
  - Fichier inexistant
  - Connexion réseau interrompue
- Tous ces problèmes sont susceptibles de générer des erreurs à l'exécution. Mais Java offre la possibilité de les gérer directement dans le code.

Exemple : Division par zéro

```
public class Division
{
    public static int divise(int a, int b)
    {
        return a/b;
    }

    public static void main(String[] args)
    {
        System.out.println("3/2="+divise(3,2));
        System.out.println("5/0="+divise(5,0)); // ligne 11
    }
}
```

Exécution

```
java Division
3/2=1
java.lang.ArithmeticException: / by zero
    at Division.divise(Division.java:5)
    at Division.main(Division.java:11)
Exception in thread "main"
```

Gérer les exception

- Le bloc try/catch : associer une action
- Throws : propager à la méthode appelante

try/catch

```
try
    {
    // le contenu de mon bloc susceptible de lever des exceptions
    }
catch (UneClasseException e)
    {
    // gestion de l'exception de type UneClasseException, au polymorphisme près
    }
catch (UneAutreClasseException e)
    {
    // idem…
    }
```

```
public class Division2
{
    public static int divise(int a, int b)
    {
    return a/b;
    }


    public static void main(String[] args)
    {
    try
        {
          System.out.println("3/2="+divise(3,2));
          System.out.println("5/0="+divise(5,0));
          System.out.println("fin du try");
        }
    catch (ArithmeticException ae)
        {
          System.out.println("On ne divise pas par zéro !");
        }
    System.out.println("fin du main()");
    }
}
```

Exécution de Division2

```
java Division2
3/2=1
On ne divise pas par zéro !
fin du main()
```

```
public class Division3
{
    public static int divise(int a, int b)
    {
    try
        {
          return a/b;
        }
    catch (ArithmeticException ae)
        {
          System.out.println("la méthode traite une exception.");
          ae.printStackTrace();
        }
    return 0;
    }
public static void main(String[] args)
{
    try {
        System.out.println("3/2="+divise(3,2));
        System.out.println("5/0="+divise(5,0));
    }
    catch (ArithmeticException ae)
        {
          System.out.println("On ne divise pas par zéro !");
        }
    System.out.println("fin du main()");
    }
}
```

Exécution de Division3

```
java Division3
3/2=1
la méthode traite une exception.
java.lang.ArithmeticException: / by zero
    at Division3.divise(Division3.java:7)
    at Division3.main(Division3.java:22)
5/0=0
fin du main()
```

Conclusion sur try/catch

- Il ne faut gérer les exception que lorsqu'on sait le faire
- Contre-exemple (cf. Division3) : gérer la division par zéro, ce qui revient à masquer l'erreur, et produire un faux résultat
- Dans de tels cas, il convient de propager l'exception plutôt que de la gérer.

Propagation avec *throws*

```
public void maMethodeQuiPropageDesExceptions(int i) throws IOException
    {
    // corps de la méthode susceptible de lever/propager des IOException
    }

public void maMethode() throws Classe1Exception, Classe2Exception
```

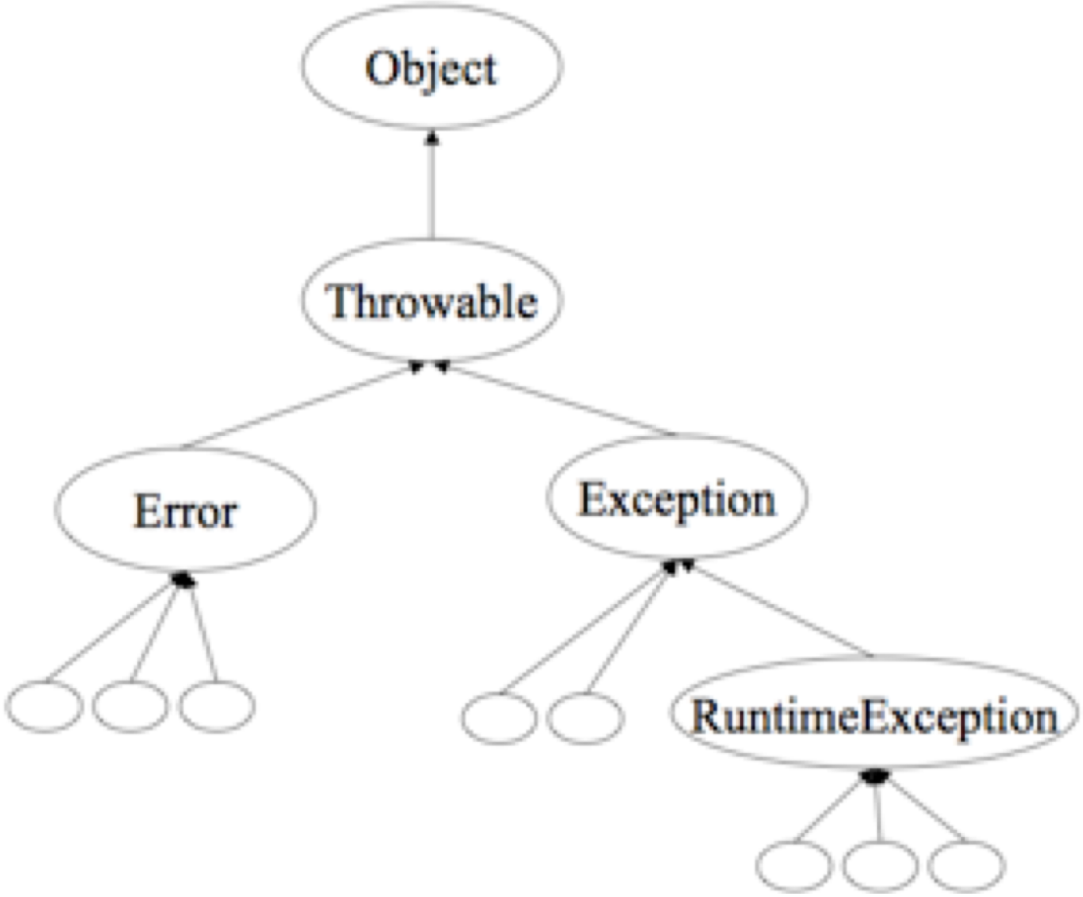
Combiner try/catch et throws…

- Un méthode peut tout à fait propager certaines exceptions (avec throws), et en gérer d'autres (avec try/catch)
- Rappel : le but n'est pas de systématiquement gérer toutes les exceptions. Il est parfois primordial qu'une méthode appelante soit informée de l'exception survenue

Conclusions sur try/catch et throws

- Mécanisme parallèle à celui, habituel, des retours de valeur (une méthode renvoie quelque chose)
- Permet de prévenir celui qui invoque la méthode que quelque chose ne va pas (mauvais paramètres, événement extérieur tel que ressource manquante, etc.)
- C'est au développeur que revient la tâche de décider à quels endroits on peut gérer quelles exceptions… et comment...

Diagramme de classes



Les deux grands types d'exceptions

- Presque chaque ligne de code est susceptible de lever une exception :
  - NullPointerException
  - ArrayIndexOutOfBoundsException
- Chaque classe d'exception peut choisir de rendre sa gestion dans le code obligatoire ou facultative:
  - Sous-classes de RuntimeException : facultatif
  - Sous-classes de Exception : obligatoire

Exceptions et polymorphisme

- L'arbre d'héritage permet la factorisation du traitement des exceptions.
- Exemple extrême : catch(Exception e) peut attraper toute exception
- Attention : bien cibler, afin d'avoir suffisamment de précisions sur le type réel d'exception
- Trop générique = trop peu d'informations

Lever (générer) une exception

- Il est possible de lever une exception à tout endroit du code
- Syntaxe : throw new UneClasseException();

Créer une classe d'exception

- Choisir un nom correspondant à sa sémantique, et se terminant par Exception
- Exemple : CercleDeRayonNulException dans le constructeur de la classe Cercle
- Choisir de rendre son traitement obligatoire ou facultatif (choix de la classe mère)

Exemple

```
class NombreNegatifException extends RuntimeException
{
    public NombreNegatifException()
    {
    super("un argument est négatif !");
    }
}

class MauvaisCoupleException extends RuntimeException
{
    public MauvaisCoupleException()
    {
    super("il s'agit d'un couple mal assorti");
    }
}
```

Utilisation de nos classes

```
public class LeveException
{
    public static double soustraction(double a, double b)
    {
    if ((a<0) || (b<0))
        throw new NombreNegatifException();
    if (a<b)
        throw new MauvaisCoupleException();
    return a-b;
    }

    public static void main(String[] args)
    {
    System.out.println(soustraction(2,3));
    }
}
```

Rappel sur RunTimeException

- Que se passe-t-il dans notre exemple si nous remplaçons RunTimeException par Exception ?

Transformer (emballer) une exception

- Problème : le contenu d'une méthode lève une exception de bas niveau (ex : nullPointerException)
- Sa sémantique ne correspond pas à la méthode (plus haut niveau)
- Solution : attraper l'exception de bas niveau, et dans son catch, lever une exception correspondant à la sémantique désirée