



Programmation orienté objet avancée

- *Gestion des exceptions* -

Françoise Dubisy

Informatique de gestion

UE 230 : Projet informatique intégré

Table des matières

1	Détecter l'erreur	3
2	Création d'une classe de type Exception	9
3	Propager l'erreur : relancer l'exception	10
4	Traiter l'erreur : capturer l'exception.....	12
5	Classes d'exception existantes	14
6	Exceptions et héritage	16
7	Plusieurs exceptions lancées/générées par la même méthode.....	17
8	Traiter plusieurs erreurs : capturer plusieurs exceptions	19
9	La clause finally	22
10	Les exceptions non contrôlées	23
11	En résumé.....	25

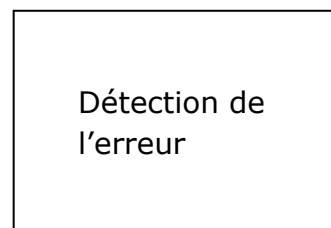
1 Détecter l'erreur

La gestion des cas d'erreurs en utilisant le mécanisme des exceptions semble a priori lourd.

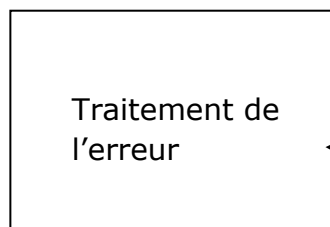
Il faut en comprendre tout l'intérêt. Pour rappel, un des points forts de la programmation orientée objet est la possibilité de réutiliser en les adaptant des composants existants. Un composant 1 écrit par un programmeur 1 peut donc être réutilisé (intégré) dans un composant 2 écrit par un autre programmeur (programmeur 2).

Le principe des exceptions est de permettre la **séparation de la détection** d'un incident (un problème ou un cas d'erreur) **de sa prise en charge**. Le cas d'erreur peut être détecté dans le composant 1 et être traité dans le composant 2 qui fait appel au composant 1. En effet, c'est le programmeur 2 qui peut décider comment doit réagir le programme lorsqu'un cas d'erreur se présente : par exemple, arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur via une boîte de dialogue, afficher un simple message sur la console ...

Composant 1 : programmeur 1



Composant 2 : programmeur 2

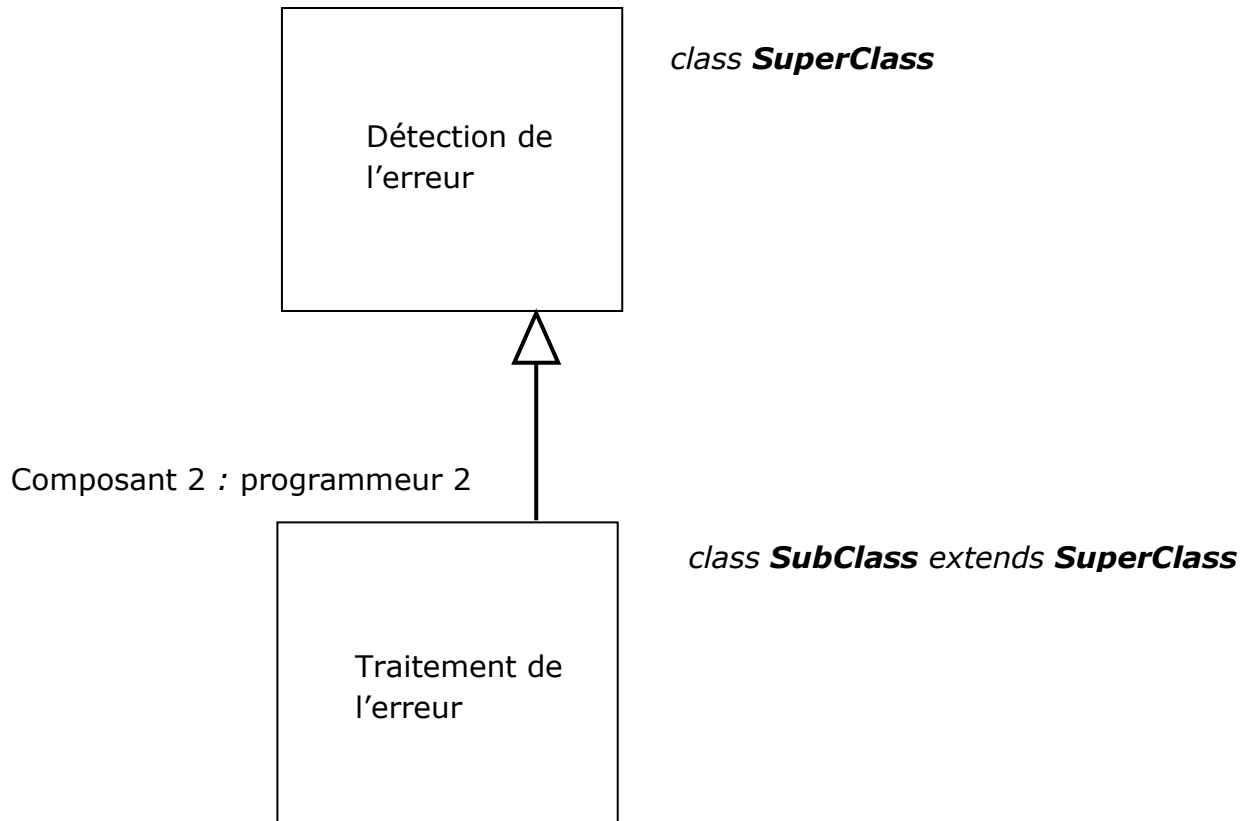


Réutilisation

Exemples de réutilisation :

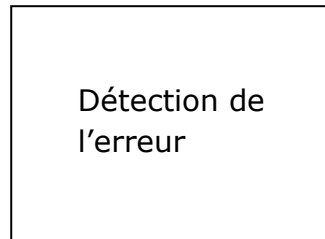
Le composant 2 est une **sous-classe** du composant 1 :

Composant 1 : programmeur 1



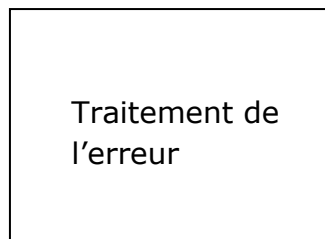
Dans l'exemple suivant, le composant 1 est une classe (*ClassX*) utilisée dans le composant 2 qui est la classe *Main*. Une occurrence (*x*) de la classe *ClassX* est créée dans la classe *Main* et des méthodes de la classe *ClassX* sont appelées sur cet objet *x*.

Composant 1 : *programmeur 1*



```
class ClassX {  
    ...methodM(...) {  
        ...  
    }  
    ...  
}
```

Composant 2 : *programmeur 2*



```
class Main {  
    ... main... {  
        ClassX x = new ClassX(...);  
        x.methodM(...);  
        ...  
    }  
}
```

Grâce au mécanisme des exceptions, le concepteur du composant 1 peut écrire le code de **détection du cas d'erreur** sans prendre de décision quant au traitement de l'erreur. Le composant 1 se contente alors d'avertir, le cas échéant, le composant appelant (composant 2) qu'une erreur a été détectée. Le programmeur du composant appelant (composant 2) peut alors **recupérer l'erreur détectée** par le composant 1 et **décider de son traitement** (*arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur...*).

Lorsqu'un cas d'erreur est détecté dans le composant 1, **un objet de type Exception est généré** par le composant 1. Cet objet peut mémoriser des informations sur l'erreur qui a eu lieu. Une classe de type Exception doit donc être créée par le programmeur 1.

Soit la classe *Person* (composant 1) écrite par le programmeur 1. Un objet de type *Person* est caractérisé entre autres par une variable d'instance de type caractère représentant son genre (variable *gender*). Partons du principe que les valeurs possibles pour le genre sont *masculin* (valeur 'm'), *féminin* (valeur 'f') et *autre* (valeur 'x'). Un cas d'erreur pourrait être détecté dans la classe *Person* si une tentative est effectuée pour affecter une valeur autre que 'm', 'f' ou 'x' à la variable d'instance *gender*.

Pour rappel, afin d'éviter toute incohérence dans la valeur affectée à cette variable d'instance, le concepteur de la classe *Person* (programmeur 1) déclare privée (**private**) la variable d'instance *gender*.

La seule possibilité pour affecter une valeur à la variable *gender* si l'on se trouve en dehors de la classe *Person* est donc d'utiliser le setter **public** correspondant (**setGender**).

Le test de **détection d'une valeur erronée** que l'on tenterait d'attribuer à la variable *gender* est à placer **dans ce setter**. Si la valeur que l'on tente d'attribuer à la variable d'instance *gender* est erronée, un objet de type *GenderException* est généré. La classe *GenderException* doit être écrite (voir point 2). Le programmeur qui détecte l'erreur et crée l'objet de type *GenderException* peut enregistrer dans cet objet des informations sur l'erreur qui a eu lieu. Ces informations seront placées dans les variables d'instance de la classe *GenderException*. On peut également préciser un message d'erreur par défaut. Dans cet exemple, deux informations seront stockées concernant l'erreur : la valeur erronée que l'on a tenté d'affecter au genre et un message par défaut décrivant l'erreur. Ces deux informations seront fournies au constructeur de *GenderException*.

```
public class Person {
    private String name;
    private char gender;

    public void setGender (char gender)    ...    {
        if (gender != 'm' && gender != 'f' && gender != 'x') {
            String message = "La valeur " + gender + " est invalide";
            throw new GenderException(gender, message);
        }
        else // NB. else inutile
            this.gender = gender;
    }
}
```

Le code à écrire pour générer un objet de type *GenderException* est :

throw new GenderException(...);

NB. To throw en anglais signifie lancer.

Le mécanisme d'exception que l'on met ainsi en place est destiné à avertir l'utilisateur du composant 1 qu'un cas d'erreur a été détecté. Or, le programmeur du composant 2 qui utilisera le composant 1 n'a **pas forcément accès au code** du composant qu'il utilise. Mais il peut par contre consulter la **documentation** accompagnant la classe qu'il utilise. Cette documentation contient entre autres la déclaration des constructeurs et méthodes déclarés **public**.

Il faut donc qu'à la seule lecture de **la déclaration d'une méthode** qu'il compte utiliser, le programmeur sache si la méthode est susceptible ou non de détecter une erreur, autrement dit si la méthode est susceptible de générer une exception, et si oui, de quel type.

Ceci explique que la déclaration d'une méthode susceptible de détecter un cas d'erreur doit contenir le mot réservé **throws** suivi du nom de la classe correspondant au type d'exception éventuellement générée.

Attention : ne pas confondre les mots réservés *throw_* et *throws_*. Le mot *throw* s'utilise dans le code de la méthode, le mot *throws* dans sa déclaration !

La déclaration complète de la méthode *setGender* est donc :

```
|| public void setGender (char gender) throws GenderException
```

Le code de la classe *Person* est donc :

```
public class Person {
    private String name;
    private char gender;

    public void setGender (char gender) throws GenderException {
        if (gender != 'm' && gender != 'f' && gender != 'x') {
            String message = "La valeur " + gender + " est invalide";
            throw new GenderException(gender, message);
        }
        this.gender = gender;
    }
}
```

Attention : **Dès qu'un cas d'erreur est détecté** et qu'un objet de type exception est créé, **le reste de la méthode est abandonné**.

Autrement dit, dès qu'une instruction contenant un *throw* est exécutée, on sort de la méthode et les instructions qui suivraient éventuellement l'instruction *throw* ne sont pas exécutées. Ce qui explique que le *else* est inutile.

2 Création d'une classe de type Exception

Une classe utilisée pour générer des exceptions est en général une **sous-classe de la classe *Exception* ou d'une de ses sous-classes**.

Toute classe d'exception peut contenir ses propres variables d'instance permettant d'enregistrer des informations sur l'erreur qui s'est produite.

Une variable d'instance qu'il serait intéressant de prévoir est une variable permettant de stocker la valeur erronée qu'on a tenté d'affecter. La mémorisation de la valeur erronée dans l'objet de type exception a pour avantage, par exemple, de permettre au programmeur qui recevra l'exception de construire un message d'erreur personnalisé, c'est-à-dire rappelant la valeur erronée. Notons qu'il faut alors prévoir le getter public associé.

Toute sous-classe d'*Exception* hérite d'une variable d'instance de type *String* qui permet de stocker un message associé à l'exception ; le contenu de cette variable peut être récupéré via la méthode héritée *getMessage()*. Le contenu de cette variable héritée est initialisé via appel au constructeur hérité.

```
public class GenderException extends Exception {  
    private char wrongGender;  
  
    public GenderException(char wrongGender, String message) {  
        super(message);  
        setWrongGender(wrongGender);  
    }  
  
    public char getWrongGender() {  
        return wrongGender;  
    }  
  
    public void setWrongGender(char wrongGender) {  
        this.wrongGender = wrongGender;  
    }  
}
```

3 Propager l'erreur : relancer l'exception

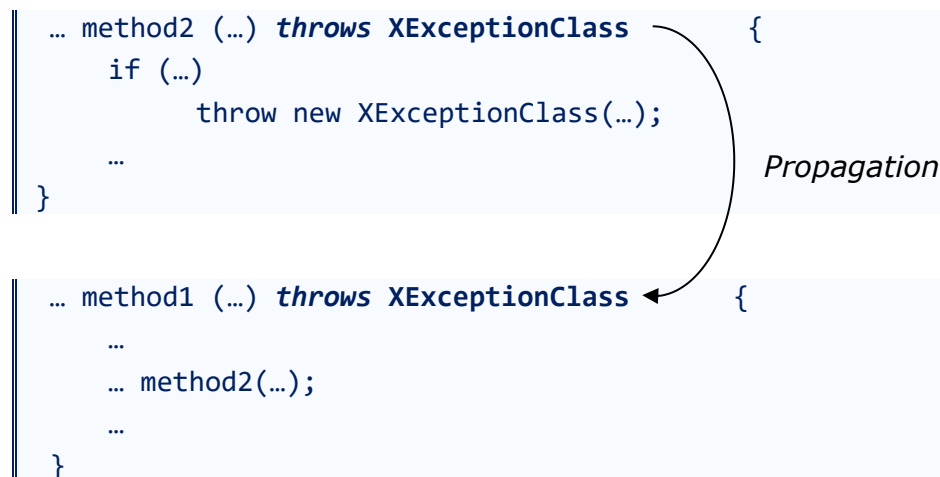
Le programmeur qui fait appel à une méthode susceptible de détecter une erreur c'est-à-dire de générer une exception (méthode qui comprend une clause *throws* dans sa déclaration) doit choisir parmi les deux solutions suivantes :

- Soit **traiter** le cas d'erreur (traiter l'exception) ;
- Soit **propager** l'exception vers le niveau appelant supérieur.

La section 4 aborde le traitement des erreurs. Nous nous concentrons dans cette section sur la propagation des exceptions.

Si le programmeur qui fait appel à une méthode susceptible de détecter une erreur ne désire **pas traiter l'erreur**, il peut décider de **reporter cette décision vers le niveau appelant**. On dit alors qu'il y a **propagation** de l'exception.

Le mécanisme de la propagation d'exception se déroule comme suit. **Une method1 qui fait appel dans son code à une method2 contenant une clause *throws* dans sa déclaration doit recopier cette clause *throws* dans sa propre déclaration.**



```
... method2 (...) throws XExceptionClass {
    if (...)
        throw new XExceptionClass(...);
    ...
}

... method1 (...) throws XExceptionClass {
    ...
    ... method2(...);
    ...
}
```

The diagram shows two code blocks. The top block represents `method2` which has a `throws XExceptionClass` clause and contains a `throw new XExceptionClass(...);` statement. The bottom block represents `method1` which also has a `throws XExceptionClass` clause and calls `method2(...)`. A curved arrow labeled "Propagation" points from the `throws XExceptionClass` clause of `method2` to the `throws XExceptionClass` clause of `method1`, indicating that the exception declaration is being propagated to the caller.

C'est le cas par exemple lorsque le concepteur de la classe *Person* (programmeur 1) a prévu que le constructeur de la classe *Person* fasse appel à la méthode *setGender* alors que celle-ci est susceptible de lancer une exception de type *GenderException*. Le concepteur de la classe *Person* peut décider de laisser à

l'appréciation du futur utilisateur de cette classe (c'est-à-dire au programmeur 2 qui appelle le constructeur) la décision du traitement de l'erreur.

Le programmeur 1 se contente alors de **recopier dans la déclaration du constructeur** la clause *throws* de la méthode *setGender*, à savoir, ***throws GenderException***.

Le code de la classe *Person* ainsi complété devient :

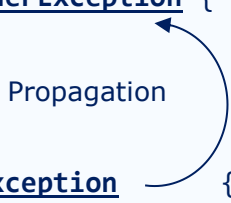
```
public class Person {
    private String name;
    private char gender;

    public Person(String name, char gender) throws GenderException {
        setName(name);
        setGender(gender);
    }

    public void setGender (char gender) throws GenderException {
        if (gender != 'm' && gender != 'f' && gender != 'x') {
            String message = "...";    // message décrivant l'erreur
            throw new GenderException(gender, message);
        }
        this.gender = gender;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String toString( ) {
        return "La Person " + name;
    }
}
```



The diagram illustrates the propagation of an exception. A curved arrow labeled "Propagation" points from the throws GenderException clause in the `setGender` method to the throws GenderException clause in the `Person` constructor, indicating that the exception is being passed up the call stack.

4 Traiter l'erreur : capturer l'exception

Si le programmeur qui fait appel à une méthode susceptible de détecter une erreur désire **traiter l'erreur**, il doit placer **l'appel à cette méthode** dans un **bloc *try*** suivi d'un **bloc *catch*** précisant la **réaction à avoir en cas d'erreur** (En anglais, "to try" signifie "essayer" et "to catch" signifie "attraper").

Le mot réservé *catch* est suivi du nom de la classe correspondant au type d'exception que l'on essaye d'attraper. On place alors dans le bloc ***catch*** le code spécifiant la façon de réagir en cas d'erreur correspondant à ce type d'exception.

Le bloc *try* peut contenir plusieurs instructions. Dès qu'une instruction génère une exception, le code du bloc *catch* correspondant est exécuté **mais le reste des instructions du bloc *try* ne sont pas exécutées**.

Prenons comme exemple la classe *Main* (composant 2 écrit par le programmeur 2).

Le code de la méthode *main* de la classe *Main* tente de créer une instance de la classe *Person*. Pour ce faire, il faut faire appel au constructeur de la classe *Person* dont la déclaration est :

```
|| public Person(String name, char gender) throws GenderException
```

Ce constructeur est susceptible de générer une exception de type *GenderException*. Son appel est donc à placer dans un bloc ***try***.

Ce bloc est suivi d'un bloc ***catch*** qui tente d'attraper une occurrence de type *GenderException*. Si une occurrence de type *GenderException* existe, c'est qu'elle a été générée lors de l'exécution de la méthode *setGender* de la classe *Person*, et donc qu'un cas d'erreur a été détecté. Si une occurrence de type *GenderException* a été générée par la méthode *setGender*, la référence de cette occurrence peut être récupérée par le programmeur, par exemple dans la variable ***genderException*** (cf *catch (GenderException ***genderException***)*).

Le traitement de l'erreur consiste ici en l'ouverture d'une boîte de dialogue qui affiche le message correspondant à l'erreur. Ce message est obtenu en appelant la méthode *getMessage* sur l'objet ***genderException*** (de type *GenderException*).

```

import javax.swing.*;          /* Si utilisation de composants swing
                                Exemple : JOptionPane (boîte de dialogue) */

public class Main {
    public static void main(String[ ] args) {
        try {
            Person jules = new Person (...);          (1)
            System.out.println(jules);                (2)
        }
        catch (GenderException genderException) {
            JOptionPane.showMessageDialog(null, genderException.getMessage(),
                                           "Erreur", JOptionPane.ERROR_MESSAGE);
        }
        (3)
        System.exit(0) ;
    }
}

```

Si l'appel au constructeur (cf (1)) génère une exception, le **reste du bloc try est abandonné** : l'instruction (2) n'est pas exécutée. Ce qui convient parfaitement, étant donné que si l'appel au constructeur a généré une exception, l'objet *jules* n'a pas été créé et donc pas initialisé, et par conséquent, l'exécution de l'instruction (2) provoquerait une erreur : il est en effet impossible d'appeler implicitement la méthode *toString* sur un objet qui n'existe pas (*jules* serait une **référence nulle**), le tout provoquerait une *Null Pointer Exception*.

Ceci explique que placer l'instruction (2) en dehors du bloc *try*, après le bloc *catch* (par exemple en (3)), constitue une erreur de programmation détectée à la compilation. Le compilateur affiche alors le message d'erreur stipulant que l'objet *jules* pourrait être **non initialisé**.

En conclusion, **tout objet que l'on tente de créer (donc d'initialiser) dans un bloc try ne peut être utilisé en dehors de ce bloc.**

5 Classes d'exception existantes

Il existe de nombreuses classes d'exception disponibles. Ces classes sont organisées en une hiérarchie d'héritage. La racine de cette hiérarchie est la classe *Exception*.

Bon nombre de méthodes dans les classes créées par les concepteurs de Java ont dans leur déclaration une clause *throws* signalant que ces méthodes sont susceptibles de générer des exceptions. L'appel à ces méthodes doit donc être placé dans un bloc *try*.

Le bloc *catch* associé peut faire appel aux méthodes **disponibles dans la classe d'exception** correspondante (cf documentation de la classe d'exception apparaissant dans la déclaration de la méthode appelée).

Parmi ces méthodes, deux paraissent intéressantes :

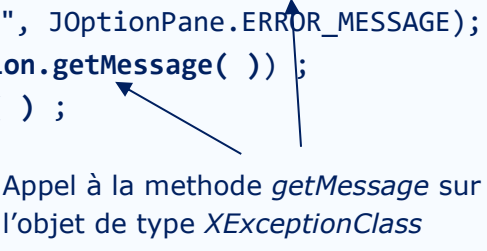
- La méthode ***getMessage*** à appeler sur un objet de type *Exception* et qui retourne une **chaîne de caractères proposant une description de l'erreur** détectée (à utiliser via l'instruction *System.out.println* ou comme message à afficher dans une boîte de dialogue par exemple) ;
- La méthode ***printStackTrace*** à appeler sur un objet de type *Exception* (ne retourne rien) qui affiche directement dans la fenêtre console le contenu de la pile des erreurs rencontrées (plus difficile à déchiffrer).

Exemple

```
public class ClassA {  
    ...  
    public ... methodM( ... ) throws XExceptionClass {  
        if ( ... )  
            throw new XExceptionClass(...);  
        ...  
    }  
}
```

```
public class Main {
    public static void main(String[ ] args) {
        try {
            ClassA a = new ClassA(...) ;
            a.methodM(...);

            ...
        }
        catch (XExceptionClass exception) {
            JOptionPane.showMessageDialog(null, exception.getMessage( ),
                                         "Erreur", JOptionPane.ERROR_MESSAGE);
            //ou : System.out.println(exception.getMessage( )) ;
            //ou : exception.printStackTrace( ) ;
        }
    }
}
```



Appel à la methode *getMessage* sur l'objet de type *XExceptionClass*

6 Exceptions et héritage

Si une sous-classe fait appel à une méthode de la super-classe qui est susceptible de générer une exception, il faut

- Soit propager l'exception (en recopiant la clause *throws* dans la déclaration de la méthode appelante) ;
- Soit la traiter (en plaçant l'appel à la méthode dans un bloc *try*).

Il en va de même pour les constructeurs. Une sous-classe fait appel dans son constructeur au constructeur de la super-classe via l'instruction *super(...)*. Si le constructeur de la super-classe est susceptible de générer une exception, il contient donc une clause *throws* dans sa déclaration. Le constructeur de la sous-classe doit soit propager l'exception, soit la traiter.

Soit la classe *Student* qui est une sous-classe de la classe *Person*. Un étudiant est en outre caractérisé par la section dans laquelle il est inscrit ainsi que l'année dans laquelle il est inscrit, par exemple en deuxième droit.

Le constructeur de la classe *Student* fait appel au constructeur de la classe *Person* via l'instruction *super(...)* .

Or, le constructeur de la classe *Person* est susceptible de générer une exception. La déclaration du constructeur de *Person* est en effet :

```
|| public Person(String name, char gender) throws GenderException
```

Le constructeur de la classe *Student* doit donc soit traiter l'erreur, soit la propager.

Dans l'exemple ci-dessous, le constructeur de la classe *Student* propage l'exception. Sa déclaration contient donc la clause : *throws GenderException*

```
public class Student extends Person {
    private String sectionName;
    private int studyYearNumber;
    public Student (String name, char gender, String section, int year)
        throws GenderException {
        super(name, gender);
        setSectionName(section);
        setstudyYearNumber(year);
    }
    ...
}
```

Appel au constructeur de la classe *Person* qui est susceptible de générer une exception de type **GenderException**

7 Plusieurs exceptions lancées/générées par la même méthode

Une même méthode peut détecter plus d'un type d'erreur. Si c'est le cas, sa déclaration doit contenir la clause **throws** suivi du nom de chacune des **classes d'exception** correspondant aux exceptions susceptibles d'être générées par la méthode :

```
|| ... methodM (...) throws XExceptionClass, YExceptionClass, ZExceptionClass
```

Rappelons que dès qu'une erreur est détectée lors de l'exécution d'une méthode et qu'un objet de type exception est par conséquent généré, le reste de la méthode n'est pas exécuté. Ceci implique qu'à l'exécution, **une méthode ne peut détecter qu'un seul cas d'erreur**. En effet, dès qu'un objet de type exception est généré, l'exécution du reste de la méthode est stoppée : une même exécution de méthode ne peut donc pas générer plus d'un objet de type exception à la fois.

À titre d'exemple, on pourrait prévoir la gestion de cas d'erreur sur l'année d'inscription d'un étudiant : les valeurs permises seraient 1, 2 ou 3. Toute tentative d'affecter une valeur autre que celles-ci serait considérée comme cas d'erreur. Le même principe que pour la gestion des valeurs affectées à la variable *gender* est appliqué ici. La variable *studyYearNumber* est déclarée **private** et un **setter public** est prévu : *setStudyYearNumber*. Ce setter étant susceptible de générer une exception de type **YearNumberException**, sa déclaration contient la clause **throws YearNumberException**. La classe *YearNumberException*, sous-classe de la classe *Exception*, doit être écrite. Le constructeur de la classe *Student* doit donc faire appel au setter *setStudyYearNumber* pour affecter la valeur reçue en argument à la variable *studyYearNumber*. La déclaration du constructeur doit donc signaler que des exceptions de deux classes différentes sont susceptibles d'être générées. Sa déclaration contient donc la clause : **throws GenderException, YearNumberException**

```
public class Student extends Person {
    private String sectionName;
    private int studyYearNumber;

    public Student(String name, char gender, String section, int year)
        throws GenderException, YearNumberException {
        super(name, gender);
        setSectionName(section);
        setStudyYearNumber(year);
    }
}
```

```

public void setStudyYearNumber(int year) throws YearNumberException {
    if (year < 1 || year > 3) {
        String message = "...";    // message décrivant l'erreur
        throw new YearNumberException(year,message);
    }
    studyYearNumber = year;
}
}

```

```

public class YearNumberException extends Exception {

    private int wrongYearNumber;

    public YearNumberException(int wrongYearNumber, String message) {
        super(message);
        setWrongYearNumber(wrongYearNumber);
    }

    public int getWrongYearNumber() {
        return wrongYearNumber;
    }

    public void setWrongYearNumber(int wrongYearNumber) {
        this.wrongYearNumber = wrongYearNumber;
    }
}

```

8 Traiter plusieurs erreurs : capturer plusieurs exceptions

Lorsqu'un programmeur fait appel à une méthode dont la déclaration contient une clause *throws* portant sur plusieurs classes de type exception, il doit comme convenu placer cet appel dans un bloc *try*. Ce bloc *try* peut être suivi de plusieurs blocs *catch* : un bloc *catch* par type d'exception à traiter. Chaque bloc *catch* contient alors le code correspondant à la façon de traiter un type d'erreur.

```
try { ...  
    ... methodM(...) ;  
    ...  
}  
catch (XExceptionClass xException) {  
    ...           // Réaction en cas d'erreur de type XExceptionClass  
}  
catch (YExceptionClass yException) {  
    ...           // Réaction en cas d'erreur de type YExceptionClass  
}  
catch (ZExceptionClass zException) {  
    ...           // Réaction en cas d'erreur de type ZExceptionClass  
}
```

Soit la classe *Student* telle que proposée à la section 7.

La classe *Main* essaye (bloc *try*) de créer une occurrence de la classe *Student*. La déclaration du constructeur de la classe *Student* est :

```
public Student (String name, char gender, String section, int year)  
    throws GenderException, YearNumberException
```

Le constructeur de la classe *Student* est donc susceptible de générer des exceptions de deux types : *GenderException* et *YearNumberException*. Si l'on désire réagir différemment à ces deux cas d'erreur, on prévoit deux blocs *catch* dans la classe *Main*. Le **premier bloc *catch*** précise comment réagir si la valeur proposée pour le **genre** est **invalid**e (on choisit dans l'exemple proposé ci-dessous d'afficher le message correspondant à l'erreur via l'instruction *System.out.println*). Le **second bloc *catch*** précise comment réagir si la valeur proposée pour l'**année** est **invalid**e (on choisit pour les besoins de l'exemple

une autre réaction que pour le genre invalide : on choisit d'afficher le message correspondant à l'erreur via une boîte de dialogue).

```
public class Main {  
    public static void main(String[ ] args) {  
        try {  
            Student jules = new Student (...);  
            System.out.println(jules);  
        }  
        catch (GenderException genderException) {  
            System.out.println(genderException) ;  
        }  
        catch (YearNumberException yearException) {  
            JOptionPane.showMessageDialog(null, yearException.getMessage( ),  
                "Erreur: année", JOptionPane.ERROR_MESSAGE);  
        }  
        System.exit(0);  
    }  
}
```

NB. Ne pas oublier l'instruction **System.exit(0)** ; en fin de programme lorsqu'on utilise des boîtes de dialogue.

Notons que si on désire réagir de la même façon lorsqu'on tente d'afficher une valeur erronée à la variable *gender* ou à la variable *studyYearNumber*, il n'est pas nécessaire de prévoir deux blocs *catch*. On pourrait par exemple afficher le message d'erreur via une boîte de dialogue, qu'il s'agisse d'une erreur sur la variable *gender* ou d'une erreur sur la variable *studyYearNumber*. Un seul bloc *catch* suffit alors :

```
|| catch (GenderException|YearNumberException exception)
```

Le titre de la boîte de dialogue est « *erreur* » et le contenu est fourni par l'appel à la méthode *getMessage* sur l'objet *exception* de type *Exception*. Le contenu du message d'erreur sera personnalisé en fonction du type d'erreur. Il s'agit de l'application du polymorphisme.

```
import javax.swing.* ;
public class Main {

    public static void main(String[ ] args) {
        try {
            Student jules = new Student (...);
            System.out.println(jules);
        }
        catch (GenderException|YearNumberException exception) {
            JOptionPane.showMessageDialog(null, exception.getMessage( ),
                                         "Erreur", JOptionPane.ERROR_MESSAGE);
        }
        System.exit(0);
    }
}
```

9 La clause **finally**

Les blocs *try* et *catch* peuvent être complétés par un bloc ***finally***. Celui-ci est placé après le dernier bloc *catch*.

Les instructions contenues dans le bloc *finally* seront exécutées qu'il y ait eu ou non une exception générée. Les instructions du bloc *finally* seront donc exécutées :

- À la suite des instructions du bloc *try*, s'il n'y a pas d'erreur,
- À la suite des instructions d'un bloc *catch*, en cas d'erreur.

```
try { ...  
    ... methodM ( ... ) ;  
    ...  
}  
catch (XExceptionClass xException) {  
    ...           // Réaction en cas d'erreur de type XExceptionClass  
}  
catch (YExceptionClass yException) {  
    ...           // Réaction en cas d'erreur de type YExceptionClass  
}  
catch (ZExceptionClass zException) {  
    ...           // Réaction en cas d'erreur de type ZExceptionClass  
}  
finally {  
    ...           // Instructions à exécuter dans tous les cas  
}
```

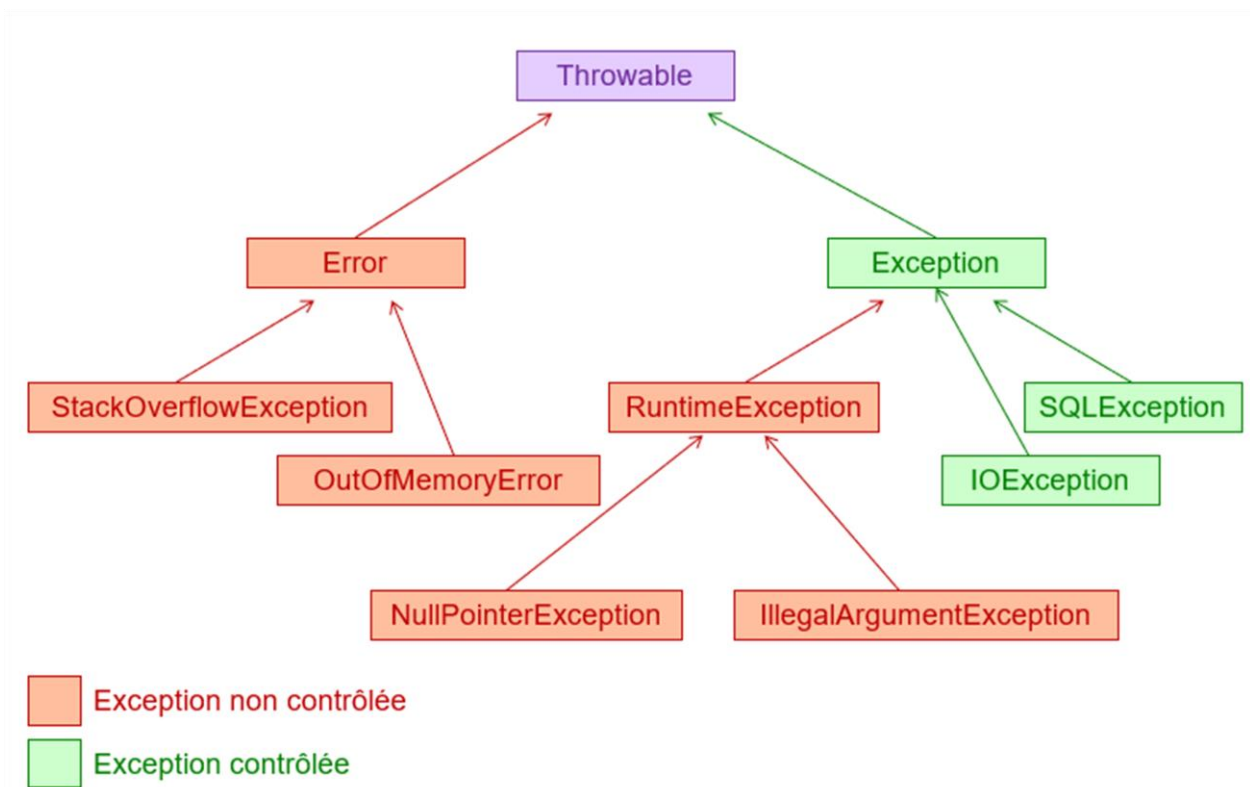
10 Les exceptions non contrôlées

Les exceptions non contrôlées sont des **sous-classes de la classe *Error* ou de la classe *RuntimeException***.

Elles peuvent ne pas être gérées, c'est-à-dire pas de *try-catch* ni de propagation.

De plus, elles ne doivent pas forcément être déclarées dans la clause *throws* des méthodes qui sont susceptibles de les lancer.

Le schéma suivant propose un extrait de la hiérarchie des exceptions :



Quand utiliser les exceptions non contrôlées ?

Ceci est laissé à l'appréciation du programmeur.

En vue de prendre la décision, on peut se poser la question suivante :

Quand l'erreur survient, est-ce que le code appelant peut réagir pour arranger la situation ?

- Si la réponse est oui, on utilise des exceptions contrôlées.
- Si la réponse est non, cela signifie qu'il peut s'agir d'une erreur de programmation (exemple : *NullPointerException*) et que le code appelant ne peut rien faire pour y remédier. On utilise alors des exceptions non contrôlées.

11 En résumé

① Le principe des exceptions est de permettre la séparation de la détection d'un incident (un problème ou un cas d'erreur) de sa prise en charge. La détection du cas d'erreur peut avoir lieu dans un composant 1 mais le traitement de ce cas d'erreur peut être réalisé dans un composant appelant le composant1.

② Pour gérer un cas d'erreur via le mécanisme des exceptions, il faut créer une **sous-classe d'une classe d'Exception**.

Exemple :

```
class XExceptionClass extends Exception
```

Lorsqu'un cas d'erreur est détecté, il faut créer une occurrence de cette sous-classe d'exception via l'instruction *throw*.

Exemple :

```
throw new XExceptionClass(...)
```

③ La méthode qui est susceptible de détecter l'erreur et donc de générer un objet de type exception doit contenir dans sa déclaration la clause ***throws* suivi du nom de la classe d'exception** correspondante.

Exemple :

```
... methodM(...) throws XExceptionClass
```


④ Dès qu'un cas d'erreur est détecté lors de l'exécution d'une méthode (et, par conséquent, qu'un objet de type exception est créé), l'exécution de la méthode est stoppée : le reste de la méthode est abandonné.

⑤ Toute méthode appelant une méthode susceptible de générer une exception doit :

- Soit propager l'exception,
- Soit traiter l'exception.

⑥ Propager l'exception consiste à reporter la décision du traitement de l'erreur vers le niveau appelant. Le mécanisme de la **propagation** d'exception se fait comme suit. Une *method1* qui fait appel dans son code à une *method2* contenant une clause *throws* dans sa déclaration doit recopier cette même clause *throws* dans sa propre déclaration :

```
... method1 (...) throws XExceptionClass {  
    ... throw new XExceptionClass(...); ...  
}  
  
... method2 (...) throws XExceptionClass {  
    ... method1(...); ...  
}
```



⑦ Traiter l'exception signifie décider de la réaction à avoir en cas d'erreur (exemples : arrêter le programme, lancer une procédure particulière, envoyer un message d'erreur à l'utilisateur...). **Pour traiter l'exception**, il faut placer l'appel à la méthode susceptible de générer un objet de type exception dans un bloc **try** et placer dans un bloc **catch** le code spécifiant la façon de réagir à l'erreur.

```
try {  
    ...  
    ... methodM(...) ; // susceptible de générer une exception de  
    ...                // type XExceptionClass  
}  
catch (XExceptionClass exception) {  
    ...                // Réaction en cas d'erreur  
}
```

⑧ Le bloc *try* peut contenir plusieurs instructions. Dès qu'une instruction génère une exception, le code du bloc *catch* correspondant est exécuté mais le **reste des instructions du bloc try ne sont pas exécutées**.

⑨ Si une **sous-classe** fait appel à une méthode de la super-classe qui est susceptible de générer une exception, il faut

- Soit propager l'exception (en recopiant la clause *throws* dans la déclaration de la méthode appelante) ;
- Soit la traiter (en plaçant l'appel à la méthode dans un bloc *try*).

①① Une même méthode peut détecter **plus d'un type d'erreur**. Si c'est le cas, sa déclaration doit contenir la clause *throws* suivi du nom de chacune des classes d'exception correspondant aux exceptions susceptibles d'être générées par la méthode :

Exemple :

```
|| ... methodM (...) throws XExceptionClass, YExceptionClass, ZExceptionClass ...
```

①① L'appel à une méthode dont la déclaration contient une clause *throws* portant sur plusieurs classes de type exception doit être placé dans un bloc *try*. Ce **bloc *try* peut être suivi de plusieurs blocs *catch*** si les réactions aux cas d'erreur sont différentes : un bloc *catch* par type d'exception à traiter. Chaque bloc *catch* contient alors le code correspondant à la façon de traiter un type d'erreur.

```
|| try { ...  
||     ... methodM ( ... ) ;  
||     ...  
|| }  
|| catch (XExceptionClass xException) {  
||     ... // Réaction en cas d'erreur de type XExceptionClass  
|| }  
|| catch (YExceptionClass yException) {  
||     ... // Réaction en cas d'erreur de type YExceptionClass  
|| }  
|| catch (ZExceptionClass zException) {  
||     ... // Réaction en cas d'erreur de type ZExceptionClass  
|| }
```

①② Un bloc *finally* peut être placé après le dernier bloc *catch*. Les instructions du bloc *finally* sont exécutées qu'il y ait ou non une exception générée.

①③ Les exceptions non contrôlées (sous-classes de la classe *Error* ou *RuntimeException*) peuvent ne pas être gérées. Elles ne doivent pas obligatoirement être déclarées dans la clause *throws* de la signature des méthodes.