

Le Langage Java

1ère année

M. Bastreggi J. Beleho P. Bettens M. Codutti
A. Hallal C. Leruste D. Nabet N. Pettiaux A. Rousseau

Haute École de Bruxelles — École Supérieure d'Informatique

Année académique 2011 / 2012

Leçon 26

Les exceptions

- Présentation
- Exceptions contrôlées
- Créer ses exceptions

Présentation

Parfois, le programme se trouve face à une **situation anormale**

- ▶ Peut être détectée par
 - la JVM (ex : division par 0)
 - ou le code Java (ex : paramètre invalide)
- ▶ Une exception est créée par le code qui a détecté le problème
- ▶ L'exception est **lancée**
- ▶ Un autre bout de code peut **attraper** l'exception
 - normalement pour résoudre le problème

Lancer une exception

Quand une situation anormale est détectée, il faut le signaler

- ▶ Créer un objet de type **Exception** (ou un fils)
- ▶ Le lancer
- ▶ La suite de la méthode n'est pas exécutée

Exemple

```
void f(int nb) {  
    if (nb<0)  
        throw new IllegalArgumentException("Nombre négatif");  
    // La suite normale ...  
}
```

Itinéraire d'une exception

Une exception remonte la «pile d'appel» jusqu'à ce qu'un bout de code dédié l'attrape.

Exemple : `main` appelle `f` qui appelle `g`

```
void g() { int nb = Integer.parseInt("Douze"); }
```

- ▶ `g` ne gère pas l'exception, elle passe à `f`
- ▶ Si `f` ne la gère pas, elle passe à `main`
- ▶ Si `main` ne la gère pas, elle passe à la JVM
- ▶ La JVM attrape tout, affiche un message complet et arrête le programme

Attraper une exception

Si on veut attraper une exception

- ▶ On englobe la partie qui peut poser problème par **try**
- ▶ La partie **catch** contient la gestion de l'exception

Exemple

```
try {  
    nb = Integer.parseInt(chaine);  
} catch (NumberFormatException ex) {  
    // gestion de l'exception  
}
```

- ▶ La méthode **parseInt** (ou une méthode appelée par elle) contient un **throw new NumberFormatException(...)**

Résumons

Déroulement quand **tout va bien**

```
main() {  
    ...  
    f()  
    ...  
}
```

```
f() {  
    ...  
    try {  
        ...  
        g()  
        ...  
    } catch (...) {  
        ... // Pas fait !  
    }  
    ...  
}
```

```
g() {  
    ...  
    h()  
    ...  
}
```

```
h() {  
    ...  
    if (test)  
        throw ...  
    ...  
}
```

Si le test est faux

⇒ le code du «catch» (en brun) n'est pas exécuté

Résumons

Déroulement quand un **problème** est détecté

```
main() {  
    ...  
    f()  
    ...  
}
```

```
f() {  
    ...  
    try {  
        ...  
        g()  
        ... // Pas fait !  
    } catch (...) {  
        ...  
    }  
    ...  
}
```

```
g() {  
    ...  
    h()  
    ... // Pas fait !  
}
```

```
h() {  
    ...  
    if (test)  
        throw ...  
    ... // Pas fait !  
}
```

Si le test est vrai

⇒ les codes en brun ne sont pas exécutés

Une exception est un objet

Exemple

```
} catch (NumberFormatException ex) {
```

- ▶ On spécifie qu'on attrape tout objet de la classe `NumberFormatException`
- ▶ Et qu'on va l'appeler `ex` dans le corps du **catch**
- ▶ On pourra poser des questions à cet objet
 - Précisions sur le problème
 - **exemple** : `ex.getMessage()`

Hiérarchie des exceptions

Cette hiérarchie des exceptions explique pourquoi on peut écrire :

```
} catch (Exception ex) {
```

- ▶ `NumberFormatException` hérite de `Exception`
- ▶ Mise en oeuvre du polymorphisme



À éviter car on attrape aussi d'**autres** exceptions. (qu'on ne saura peut-être pas traiter)

Attraper plusieurs exceptions

On peut attraper plusieurs types d'exceptions

Exemple

```
try {  
    nb = Integer.parseInt(chaine);  
} catch (NumberFormatException ex) {  
    // La chaine ne contient pas un int  
} catch (Exception ex) {  
    // Autre problème  
}
```

- On exécute le premier **catch** qui est en adéquation avec l'exception lancée

Attraper plusieurs exceptions

L'ordre à son importance

Exemple

```
try {  
    nb = Integer.parseInt(chaine);  
} catch (Exception ex) {  
    // Autre problème  
} catch (NumberFormatException ex) {  
    // La chaine ne contient pas un int  
}
```

- ▶ En cas de `NumberFormatException` c'est la partie `Exception` qui est activée
⇒ Interdit par le compilateur

Exemple complet

```
public class Outil
{
    public static void afficherLigne (int nb) {
        if (nb<1 || nb >80)
            throw new IllegalArgumentException(" taille entre 1 et 80!");
        for( int i=1; i<=nb; i++) {
            System.out. print ( '—');
        }
        System.out. println ();
    }

    public static void usage() {
        System.out. println ( "usage: java Test nb (1 à 80)");
        System.exit (1);
    }
}
```

Exemple complet

```
public class Test
{
    public static void main(String[] args) {
        int nb = 0;
        if (args.length != 1) Outil.usage();
        try {
            nb = Integer.parseInt(args[0]);
        } catch (NumberFormatException ex) {
            Outil.usage();
        }
        try {
            Outil.afficherLigne(nb);
        } catch (IllegalArgumentException ex) {
            Outil.usage();
        }
    }
}
```

Exemple complet

Ou bien (code moins éclaté)

```
public class Test
{
    public static void main(String[] args) {
        if (args.length != 1) Outil.usage();
        try {
            int nb = Integer.parseInt(args[0]);
            Outil.afficherLigne(nb);
        } catch (NumberFormatException ex) {
            Outil.usage();
        } catch (IllegalArgumentException ex) {
            Outil.usage();
        }
    }
}
```

Exemple complet

Ou encore (dangereux : on capture toutes les erreurs)

```
public class Test
{
    public static void main(String[] args) {
        if (args.length != 1) Outil.usage();
        try {
            int nb = Integer.parseInt(args[0]);
            Outil.afficherLigne(nb);
        } catch (Exception ex) {
            Outil.usage();
        }
    }
}
```


Exemple complet



En Java 7, on peut combiner des exceptions

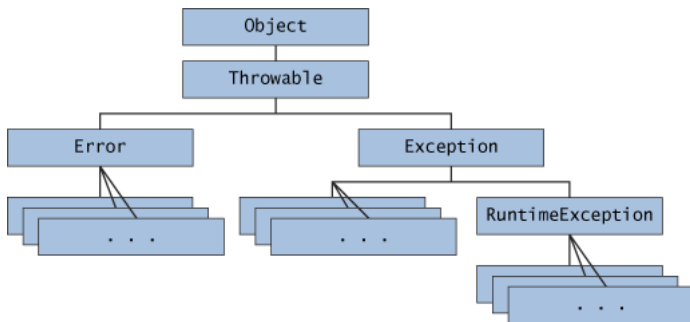
```
public class Test
{
    public static void main(String[] args) {
        if (args.length != 1) Outil.usage();
        try {
            int nb = Integer.parseInt(args[0]);
            Outil.afficherLigne(nb);
        } catch (NumberFormatException | IllegalArgumentException ex) {
            Outil.usage();
        }
    }
}
```

Exceptions contrôlées

Philosophie de Java : obliger le programmeur à coder proprement

- ▶ \Rightarrow Toute exception devrait être gérée
- ▶ Mais beaucoup d'instructions peuvent provoquer une exception (ex : [ArrayIndexOutOfBoundsException](#))
- ▶ On ne peut pas mettre des **try** partout
- ▶ \Rightarrow Java différencie les exceptions

Différentes sortes d'exceptions



source : <http://java.sun.com/docs/books/tutorial/essential/exceptions/index.html>

`java.lang.Throwable` : type commun à toutes les exceptions

Différentes sortes d'exceptions

Exception

- ▶ Toutes les exceptions **contrôlées** par le compilateur
- ▶ On doit **explicitement** les gérer ou les laisser passer. Sauf pour...

RuntimeException

- ▶ Sous-ensemble de [Exception](#)
- ▶ Pas d'obligation de les traiter

Error

- ▶ Liées au dysfonctionnement de la machine virtuelle
- ▶ Il est déconseillé de les traiter

Exceptions contrôlées

Si une méthode **peut** lancer une exception **contrôlée**

- ▶ Elle doit le déclarer dans la signature

```
void f() throws IOException {  
    ...  
    if (...)  
        throw new IOException("...");  
    ...  
}
```

- ▶ **Remarque** : ne pas confondre **throw** et **throws**

Exceptions contrôlées I

Quand on appelle une méthode qui **peut** lancer une exception **contrôlée**

- ▶ On est **obligé** de la gérer

```
void g() {  
    ...  
    try {  
        f();  
    } catch( IOException ex ) {  
        // gérer l'exception  
    }  
    ...  
}
```

Exceptions contrôlées II

- ▶ **ou** de déclarer qu'on la lance (qu'on la laisse passer en fait)

```
void g() throws IOException {  
    ...  
    f();  
    ...  
}
```

- ▶ Tout cela est vérifié par le compilateur

Créer ses propres exceptions

Revient à créer une classe qui hérite de `Exception` (ou `RuntimeException`)

Exemple :

```
public class NombreNégatifException extends Exception {  
    public NombreNégatifException(String s) {  
        super(s);  
    }  
}
```

- ▶ **extends** `Exception` indique que la classe hérite de `Exception`
- ▶ **super(s)** est lié à l'héritage.
On appelle le constructeur du parent

Créer ses propres exceptions

Exemple : Lancer et capturer une exception propre

```
public class Test {  
    public void afficherLignes (int nb) throws NombreNégatifException {  
        if (nb<0)  
            throw new NombreNégatifException("ça coince");  
        // la suite normale ...  
    }  
  
    public static void main(String[] args) {  
        Test test = new Test();  
        try {  
            test.afficherLignes (args [0]);  
        } catch (NombreNégatifException ex) {  
            System.out. println (ex.getMessage());  
        }  
    }  
}
```

Exemple complet

```
public class Groupe {  
  
    private Etudiant[] étudiants;  
    private int nbEtudiants;  
  
    public Groupe(int tailleGroupe) {  
        étudiants = new Etudiant[tailleGroupe];  
        nbEtudiants = 0;  
    }  
  
    public void ajouter(Etudiant étudiant)  
        throws DépassementCapacitéException {  
        if (nbEtudiants == étudiants.length)  
            throw new DépassementCapacitéException("Plus de place!");  
  
        étudiants[nbEtudiants] = étudiant;  
        nbEtudiants++;  
    }  
  
    // Les autres méthodes ici  
}
```

Exemple complet

```
public class DépassementCapacitéException extends Exception {  
    public DépassementCapacitéException(String s) {  
        super(s);  
    }  
}
```

Remarque : Si on n'avait pas fait le test de dépassement de capacité

- ▶ **ajouter** aurait lancé une `IndexOutOfBoundsException`
- ▶ Pas clair pour l'utilisateur (qui n'a pas connaissance de l'implémentation)

La clause «finally»

- ▶ La clause **finally** est toujours exécutée
- ▶ À la fin du **try** ou après le **catch**
 - Même en présence d'un **return**
 - ou si une exception est lancée dans le **catch**
- ▶ Permet d'indiquer un code qui doit toujours être exécuté
- ▶ Rarement utilisé (par ex : pour fermer une ressource externe comme un fichier)

Crédits

Ce document a été produit avec les outils suivants

- ▶ La distribution **Ubuntu** du système d'exploitation **Linux**
- ▶ **LaTeX** comme système d'édition
- ▶ La classe **Beamer** pour les transparents
- ▶ Les packages **listings**, **fancyvrb**, ...
- ▶ Les outils **make**, **rubber**, **pdfnup**, ...