INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Université Polytechnique HAUTS-DE-FRANCE

Cours de JAVA

Gestion d'Erreurs & Entrées-Sorties

Emmanuel ADAM

INSA Hauts-de-France

Gestion d'exceptions

Les exceptions

- La gestion des exceptions est très importante, voire primordiale, dans tous systèmes informatiques.
- Eviter les applications qui plantent sans information

Les exceptions (2)

- Les exceptions sont des instances des classes héritant des classes
  - java.lang.Error (pour des erreurs graves quasi impossibles à gérer : plus de mémoire, classe manquante, ...)
  - java.lang.Exception (pour des exceptions attendues sinon probables pouvant être gérée : débordement d'un tableau, erreur de calcul, ...).
- Ces deux classes implémentent l'interface Throwable

Les exceptions (3)

- La classe Exception possède deux constructeurs Exception() et Exception (String msg)).
-
- Elle hérite de l'interface Throwable de :
  - getMessage qui permet de récupérer le message de l'exception s'il existe.
  - toString, qui retourne la classe et le message sous forme de chaîne.
  - printStackTrace, qui fait appel à toString, mais qui en plus indique l'endroit du programme où a été levée l'exception.

Capture d'une exception

Pour récupérer, ou attraper, une exception :

```
…
try
   {…. //endroit où pourrait apparaître une exception }
catch (typeException1  e)
   { …. // traitement de l'exception de type 1 }
catch (typeException2 | typeException3  e)
   { …. // traitement de l'exception de type 2 ou 3 }
finally
   { …. // dans tous les cas, passer par ici }
…
```

Exemple (1)

```
int[] a = {1,2,3,4,5,6,7,8,9,10};
int[] b = {10,9,8,7,6,5,0,3,2,1};
int []c = new int[10];
for(int i=0; i<=10; i++) {
 try {
  c[i] = a[i] / b[i];
  System.out.println("c["+i+"] = " + c[i]);
 }
 catch(Exception e) {
  System.out.println("division par 0 ...");
  c[i] = 0;
  System.out.println("dans c                    c[i]);

  }
}
```

Exception levée

Exception attrapée et gérée

Reprise du programme

Des classes d'Exception

```
ClassNotFoundException, CloneNotSupportedException,
IllegalAccessException, .InstantiationException,
InterruptedException, NoSuchFieldException,
NoSuchMethodException, RuntimeException,
ArithmeticException, ArrayStoreException,
ClassCastException, IllegalArgumentException,
IllegalThreadStateException, NumberFormatException,
IllegalMonitorStateException, IllegalStateException,
IndexOutOfBoundsException,
ArrayIndexOutOfBoundsException,
StringIndexOutOfBoundsException,
NegativeArraySizeException, NullPointerException,
SecurityException, UnsupportedOperationException
```

Exemple (2)

```
int[] a = {1,2,3,4,5,6,7,8,9,10};
int[] b = {10,9,8,7,6,5,0,3,2,1};
int []c = new int[10];

for(int i=0; i<=10; i++) {

   try {
      c[i] = a[i] / b[i];
      System.out.println("c["+i+"] = " + c[i]);
   }

   catch(ArithmeticException e) {
      System.out.println("division par 0 ...");
      c[i] = 0;
      System.out.println("dans ce cas c["+i+"] = " + c[i]);
   }

   catch(Exception e) {
      System.out.println("... erreur imprevue ...\n" + e);
   }
}
```

Propagation d'une exception

- Si une exception n'est pas gérée dans une procédure, elle se propage hors de la procédure,
- Il faut donc que cette procédure soit capable de propager l'exception (utilisation du mot clé **throws**),
- Si à la fin du programme, une exception n 'a pas été attrapée, la pile des méthodes traversées par l'exception est affichée.

Exemple (3)

```
public void diviser() throws Exception{
  for(int i=0; i<=10; i++) {
    c[i] = a[i] / b[i];
    System.out.println("c[" + i + "] = " + c[i]);
  }
}

void test() {
  try { diviser(); }

 catch(ArithmeticException e) {
    System.out.println("division par 0 ...");
  }

 catch(Exception e) {
    System.out.println("... erreur impre
  }

  System.out.println("suite du prog");
}
```

Exception propagée

Exception levée

Exception attrapée et gérée
Mais tableau non parcouru totalement ! → boucle cassée

Définir son exception

Il est possible de définir une exception en la faisant dériver de la classe Exception :

```
class MonExceptionDivision extends Exception
{
    public String toString()
        {
            return "Problème de division";
        }
}
```

Le mot clé **throw** permet de lancer son exception.

Exemple (4)

```
public void diviser() throws Exception{
        for(int i=0; i<=10; i++) {
            if (b[i] == 0) throw new MonExceptionDivision();
            c[i] = a[i] / b[i];
            System.out.println("c[" + i + "] = " + c[i]);
        }
    }


    void test() {

        try {
            diviser();
        }

        catch(Exception e) {
            System.out.println("... erre
        }

        System.out.println("suite du prog");
    }
```

Exception propagée

Exception lancée

Exception attrapée et gérée

Les Entrées-Sorties

les package java.io.*/ java.nio.*

La classe Java.lang.System

- Utilisation des classes et interfaces du package java.io
- Il existe trois flux (flots) :
  - **System.in** : flux d'entrée standard,
  - **System.out** : flux de sortie standard,
  - **System.err** : flux de sortie pour message d'erreur.
- Les entrées sorties peuvent générer une exception de type **IOException** à gérer

Fonctionnement de l'E/S en Java

- Java procède par étape pour l'E/S dans un fichier, une passerelle réseau, …
- Il faut tout d'abord définir un flux d'entrée ou de sortie vers un objet source ou de destination
- Puis, il faut utiliser un objet d'entrée ou de sortie dédié au type de données à lire/envoyer

Gestion de fichiers

- La classe **File** permet d'accéder aux caractéristiques des fichiers et répertoires.
- Trois constructeurs :
  - `File(File dir, String name);`
  - `File(String path, String name);`
  - `File(String path).`
- Les différentes méthodes sont :
  - `canRead() ; canWrite() ; delete() ; equals(Object obj) ; exists() ; getAbsolutePath() ; getCanonicalPath() ; getName() ; getParent() ; getPath() ; hashCode() ; isAbsolute() ; isDirectory() ; isFile() ; lastModified() ; length() ; list(FilenameFilter filter) ; list() ; mkdir() ; mkdirs() ; renameTo(File dest) ; toString().`

Exemple d'utilisation de 'File'

```
import java.io.*;

public class ChercheFichier {
   public static void main(String args[]) {
      String nomFichier = "ChercheFichier.java" ;
      File f = new File(nomFichier);

      if (f.exists()) {
         String nomRep = f.getAbsolutePath(); //  retourne le répertoire et le fichier
         nomRep = nomRep.substring(0, nomRep.lastIndexOf("\\"));
         System.out.println( nomFichier + " est dans le repertoire  " + nomRep);
         System.out.println(" droits en lecture, ecriture : " + f.canRead() + "," + f.canWrite() );
      }
   }
}
```

Flux d'entrées

- **InputStream** dont dérivent les classes `ByteArrayInputStream`, **`FileInputStream`**, `FilterInputStream`, **`ObjectInputStream`**, `PipedInputStream`, **`SequenceInputStream`** et `StringBufferInputStream`
- **Reader** (orientée caractères) dont dérivent les classes **`BufferedReader`**, `LineNumberReader`, `CharArrayReader`, `FilterReader`, `PushbackReader`, `InputStreamReader`, **`FileReader`**, `PipedReader`, **`StringReader`**
- **Scanner** (orientée caractères) qui permet de lire une entrée texte et également d'utiliser un parseur

Flux de sorties

- **`OutputStream`** dont dérivent les classes

  `ByteArrayOutputStream`, **`FileOutputStream`**,

  `FilterOutputStream`, `ObjectOutputStream`, `PipedOutputStream`

- **`Writer`** (orientée caractères) dont dérivent les classes

  `BufferedWriter`, `CharArrayWriter`, `FilterWriter`,

  `OutputStreamWriter`, `PipedWriter`, **`PrintWriter`**, **`StringWriter`**

Ecriture filtrée dans un fichier ‘DataOutputStream 1/2’

Ecriture de données de type primitive,

```
public static void main(String args[])
{
  File fichier = new File("data.bin");
  FileOutputStream fos = null;
  DataOutputStream dos = null;
  try {
   FileOutputStream fos = new FileOutputStream(fichier);
   DataOutputStream dos = new DataOutputStream(fos);
   dos.writeBoolean(true); dos.writeInt(17); dos.writeDouble(17.89);
   dos.writeBoolean(false); dos.writeInt(16); dos.writeDouble(16.84);
    }
  catch (Exception e) {
      System.out.println("erreur dans le stockage :" + e) ;}
  finally  {   //fermer les flux ouverts...
     dos.close();
     fos.close();}
}
```

Ecriture filtrée dans un fichier ‘DataOutputStream 2/2’

Notation possible depuis java 8,

```
public static void main(String args[])
{
    File fichier = new File("data.bin");
   //notation appellée try-with… ferme automatiquement les flux ouverts...
    try (
        FileOutputStream fos = new FileOutputStream(fichier);
        DataOutputStream dos = new DataOutputStream(fos);)
    {
        dos.writeBoolean(true); dos.writeInt(17); dos.writeDouble(17.89);
        dos.writeBoolean(false); dos.writeInt(16); dos.writeDouble(16.84);

     }
     catch (Exception e) {System.out.println("erreur dans le stockage :" + e) ;}
}
```

Lecture filtrée d'un fichier 'DataInputStream'

Lecture de données typées, *(depuis java 8)*

```
...
File fichier = new File("data.bin");
if(fichier.exists()) {
  try (
        FileInputStream fis = new FileInputStream(fichier);
        DataInputStream dis = new DataInputStream(fis);) {
      System.out.println(dis.readBoolean() + "," + dis.readInt() + "," + dis.readDouble());
      System.out.println(dis.readBoolean() + "," + dis.readInt() + "," + dis.readDouble());
    }
    catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}
}
```

Lecture en séquence de fichiers ‘SequenceInputStream’

SequenceInputStream(Enumeration e) :
permet de lire les InputStream de l’énumération en séquence
SequenceInputStream(InputStream s1, InputStream s2) :
permet de lire deux InputStream en séquence

```
public static void main(String args[]) {
      File fichier1 = new File("data1.txt"); File fichier2 = new File("data2.txt");
      if(fichier1.exists() && fichier2.exists())
      {
            try ( FileInputStream fis1 = new FileInputStream(fichier1);
                  FileInputStream fis2 = new FileInputStream(fichier2);
                  SequenceInputStream sis = new SequenceInputStream(fis1, fis2);)
                {
                  String chaine = "";
                  int caractere = 0;
                  while ((caractere = sis.read()) !=-1)    chaine += (char)caractere;
                  System.out.println(chaine) ;
            }
catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}
  }
```

l’é

Flots de caractères

- Notion de flot de caractères (characters Streams) :
  - On ne travaille plus uniquement sur des bytes, mais sur des caractères
  - Apparition de 2 classes abstraites : Writer et Reader
  - Pour les fichiers, des classes :
    - FileReader, **Scanner** <-> FileInputStream
    - FileWriter <-> FileOutputStream
  - Echange facile de données avec accents (unicode)

Entrée de flots de caractères

Flot de caractères	Description
Reader	Classe Abstraite pour flot d'entrée (InputStream)
BufferedReader	Bufférise les entrées ligne par ligne
InputStreamReader	Transforme un flot de byte et car. Unicode
FileReader	Transforme des bytes lus depuis un fichier en car.
FilterReader	Classe abstraite pour filtrer les caractères d'entrées
PipedReader	Lecture depuis PipeWriter
StringReader	Lit depuis une chaîne de caractères
java.util.Scanner	**Lit depuis un fichier, permet la conversion de données et l'utilisation de pattern**

Sortie de flots de caractères

Flot de caractères	Description
Writer	Classe Abstraite pour flot de sortie (OutputStream)
BufferedWriter	Bufférise les sorties ligne par ligne
CharArrayWriter	Ecrit un tableau de caractères
InputStreamWriter	Transforme un flot de car. Unicode en flot de bytes
FileWriter	Transforme un fichier de car. en flot de bytes
PrintWriter	Ecrit des valeurs et des objs dans un Writer
PipedWriter	Ecrit dans un PipeWriter
StringWriter	Ecrit dans une chaîne de caractères

Ecriture de fichier texte (FileWriter)

```
public static void main(String args[])
{
    File f = new File("./files/", "Essai.txt");
    try (FileWriter fw = new FileWriter(f);
          PrintWriter pw = new PrintWriter(fw);)
        {
          // ou PrintWriter pw = new PrintWriter(new FileWriter(f));
          pw.println("bonjour");
          pw.println("ceci est le mode d'emploi.");
    }
   catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}
}
```

Lecture de fichier texte (Scanner)

```
public static void main(String args[])
{
    String chaine = "";
    File f = new File("./files/", "Essai.txt");
    try ( Scanner sc = new Scanner(f);) {
        while (sc.hasNextLine())
        {
            chaine = sc.nextLine();
            System.out.println(chaine);
        }
    }
   catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}

}
```

Saisie clavier par Scanner

```
public static void main(String args[])
{
    String chaine = "";
    System.out.print("entrez quelques mots : ");

    try ( Scanner sc = new Scanner(System.in);)
    {
          chaine = sc.nextLine();
    }
    catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}

   System.out.println(chaine);
}
```

La sérialisation

- Signifie persistance,
- Pour éviter la perte des données à la fin de l'exécution
- Si un objet contient un autre objet, ce dernier est aussi sérialisé, et on obtient
  un arbre de sérialisation

Sérialisation (2)

- Les objets **transients** (*transitoires,éphémères*)
  - Ce sont des objets qui *ne peuvent pas être sérialisés* (processus, flux de données)
  - Si ils sont définis en tant qu'attributs dans un objet sérialisé, il faut indiquer qu'ils sont transitoires
  - idem si on ne veut pas que certains attributs d'un objet soient sérialisés

Syntaxe pour la sérialisation

- La classe doit implémenter l'interface serializable (pas de méthode à implémenter)
- Exemple :

```
import java.io.*;

class MaClasse implements Serializable
{
    public transient Thread mathread;
    private String nom;
    private int total;
    ...
```

Sauvegarde d'un objet

```
public static void main(String args[])
{
    Personne p1 = new Personne("Mérovée");
    Personne p2 = new Personne("Childeric");
    File f = new File("C:\\Progjava\\fichier", "Essai.obj");
    try ( FileOutputStream fos = new FileOutputStream(f);
          ObjectOutputStream oos = new ObjectOutputStream(fos);)
    {
          oos.writeObject(p1);
          oos.writeObject(p2);
    }
   catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}
}
```

Lecture d'un objet persistant

```
public static void main(String args[])
{
    File f = new File("C:\\Progjava\\fichier", "Essai.obj");
    try ( FileInputStream fis = new FileInputStream(f);
          ObjectInputStream ois = new ObjectInputStream(fis);)
    {
          Personne p1 = (Personne) ois.readObject();
          Personne p2 = (Personne) ois.readObject();
    }
   catch (Exception e) {System.out.println("erreur dans la lecture:" + e) ;}
}
```