

Cours
Programmation Orientée Objet 2
Pour
ING 2

Chap 02:
Les exceptions

MEKAHLIA Fatma Zohra LAKRID
Maître de Conférences Classe B

Laboratoire de Modélisation, Vérification et Evaluation des Performances des systèmes complexes (MOVEP)
Bureau 123

PLAN

- Introduction.
- Définition de la notion d'exception.
- Les différents types d'exceptions.
- Hiérarchie des classe d'exception.
- Gestion des exceptions (bloc try .. catch).
- Plusieurs exceptions dans un bloc try..catch..finally.
- Déclenchement manuel d'une exception prédéfinie.
- Définir une classe d'exception.
- Utiliser une classe d'exception définie par le programmeur.
- Capturer une exception définie par le programmeur.

INTRODUCTION

- Il vous est sûrement déjà arrivé d'avoir un gros message affiché en rouge dans la console d'eclipse : ceci a été généré par une exception **qui n'a pas été capturée**.
- La gestion des exceptions s'appelle aussi **la capture d'exception** !
- Donc, un programme Java doit traiter des **situations inattendues** (exceptionnelles) en dehors de sa tâche principale.

Problems Declaration Console

<terminated> TestSansException [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (14 déc. 2021 à 13:11:16)

Exception in thread "main" java.lang.ArithmeticException: / by zero
at TestSansException.main(TestSansException.java:7)

DÉFINITION DE LA NOTION D'EXCEPTION


- Une **exception** est un événement qui se produit lors de l'exécution d'un programme et qui perturbe le flux normal des instructions du programme.
- Le **principe de la gestion** consiste à repérer un morceau de code qui pourrait générer une exception (une division par zéro, par exemple):
 - 1) **De détecter et traiter** ces erreurs en utilisant les trois mots clés (try, catch et finally).
 - 2) Ou bien, **de les lever ou les propager** en utilisant (throw et throws).
- Et dans le but d'afficher un message personnalisé et de continuer le traitement de programme principal.

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

exception de la division par zéro:

ArithmeticException



```
2 public class TestSansException {
3
4 public static void main(String[] args) {
5 int i = 10;
6 int j = 0;
7 System.out.println("résultat = " + (i / j));
8 System.out.println("et moi !!je ne serai jamais exécutée !");
9 }
0 }
```

Problems Declaration Console

<terminated> TestSansException [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (14 déc. 2021 à 13:11:16)

Exception in thread "main" [java.lang.ArithmeticException](#): / by zero
at TestSansException.main([TestSansException.java:7](#))

RuntimeException:

Exception de la division par zéro: **ArithmeticException**

- lorsque l'exception a été levée, le **programme s'est arrêté !**
- Lorsqu'un événement que la JVM ne sait pas gérer, une exception est levée (**ex : division par zéro**) !
- le type de l'exception qui a été déclenchée (levée) est **ArithmeticException**.
- On dit, qu'une division par zéro est une ArithmeticException.
- Donc, nous allons pouvoir **la capturer**, et réaliser **un traitement** en conséquence.

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

exception de débordement d'indices dans un tableau: **ArrayIndexOutOfBoundsException**

```
public class TestException1 {  
    public static void main (String[] args) {  
        int tabEnt[] = { 1, 2, 3, 4 };  
  
        // Exception de type java.lang.ArrayIndexOutOfBoundsException  
        // non contrôlée par le programme; erreur d'exécution  
        tabEnt [4] = 0; // erreur et arrêt  
  
        System.out.println ("Fin du main"); // le message n'apparaît pas  
    } // main  
} // class TestException1
```

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

exception de débordement d'indices dans un tableau: **ArrayIndexOutOfBoundsException**

- L'exécution du programme ci-dessus provoque un message d'erreur indiquant une exception du type **ArrayIndexOutOfBoundsException**, et l'arrêt du programme. Le tableau de 4 entiers `tabEnt` a des indices de 0 à 3. L'accès à la valeur d'indice 5 provoque une exception non captée.

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

Accès au contenu d'un objet null :

NullPointerException

Exemple:

- String s1 = null;
- System.out.println (s1.charAt (3)); // exception de type **NullPointerException**.
- **s1** est une référence null sur un (futur) objet de type String. On ne peut pas accéder au troisième caractère de l'objet String référencé par null.

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

Transtypage non cohérent de deux objets de classes différentes:

ClassCastException

Exemple:

- Object p = new Point();
- Color coul = (Color) p; // exception

ClassCastException : p n'est pas de la classe Color

DIFFÉRENTS TYPES D'EXCEPTIONS:

RUNTIMEEXCEPTION

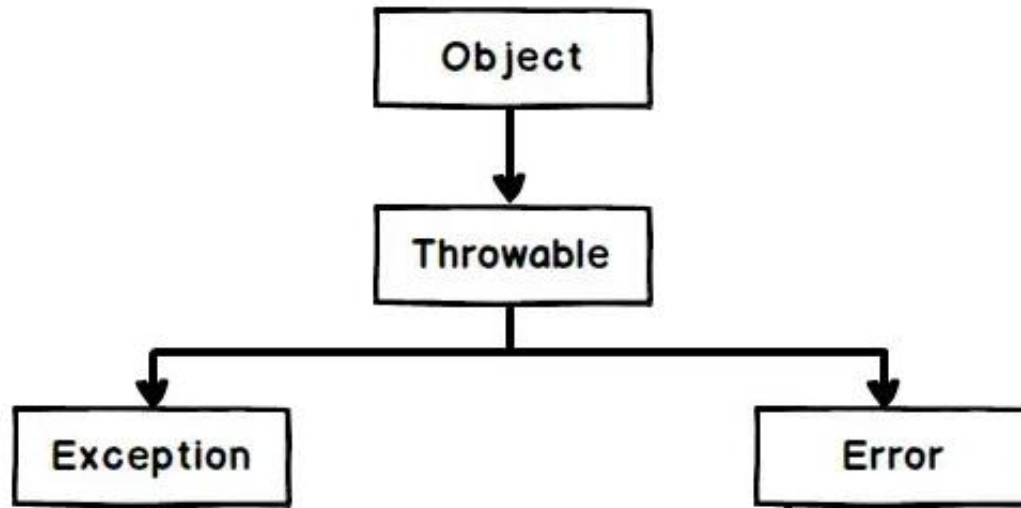
Lecture de nombres erronés:

NumberFormatException

- Exemple: la lecture d'un nombre entier depuis un champ de texte.
- `int n1 = Integer.parseInt("231");` // exception car la chaîne "231" n'est pas un entier
- `System.out.println("n1 : " + n1);`

HIÉRARCHIE DES CLASSE D'EXCEPTION

- En Java, une erreur (exception) ne provoque pas l'arrêt brutal du programme mais **la création d'un objet Throwable** . Cette classe descend directement de la classe Object : c'est la classe de base pour le traitement des exceptions en Java.



HIÉRARCHIE DES CLASSE D'EXCEPTION

Cette classe possède deux constructeurs :

- **Throwable();**
- **Throwable(String);**

Les principales méthodes de la classe Throwable sont :

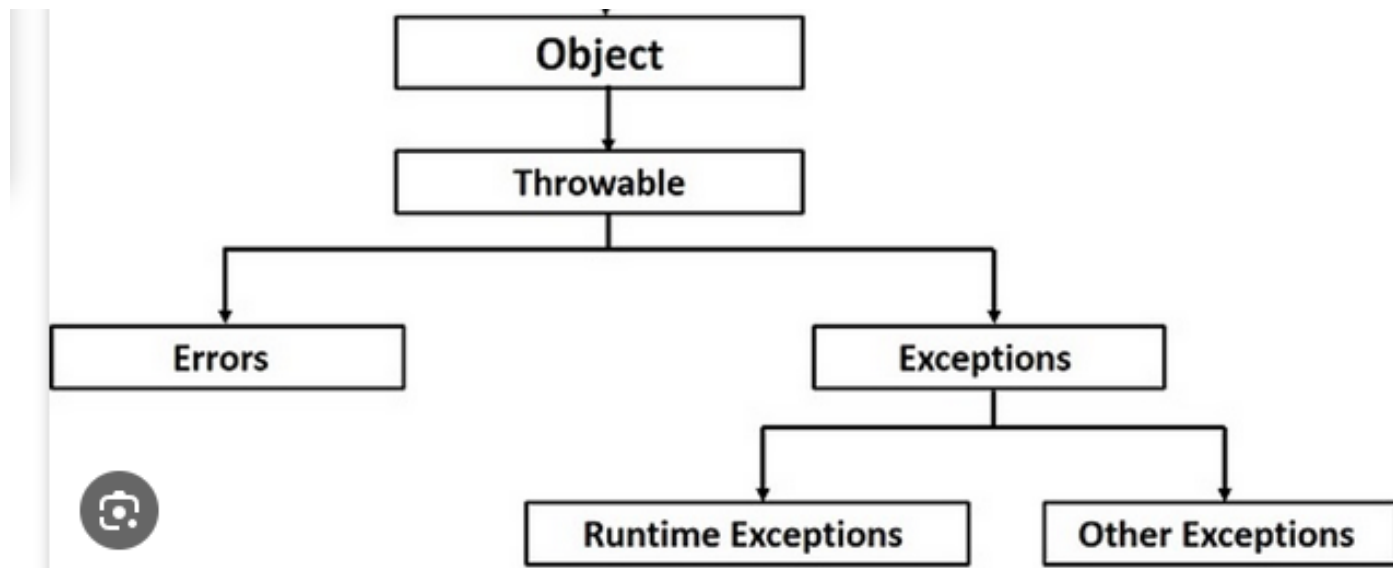
- **String getMessage();** // lecture du message
- **void printStackTrace();** /* affiche l'exception et l'état de la pile d'exécution au moment de son appel*/
- **void printStackTrace(PrintStream s);** /* Idem mais envoie le résultat dans un flux */

HIÉRARCHIE DES CLASSE D'EXCEPTION

- **Exception** et **Error** sont des sous-classes de la classe Throwable.
- La sous-classe **Error** est formée des exceptions qui ne sont pas considérées comme récupérables. Elle est utilisée par Java Virtual Machine (**JVM**) et traite les exceptions qui ne sont pas *liée au développeur* c.-à-d. ce sont des choses liées soit au **matériel** soit aux **d'autres trucs** par exemple **réseaux**.
- **Exemple:** dans un programme Java qui veut stocker des données sur le disque, mais le disque est plein ! **OutOfMemoryError**

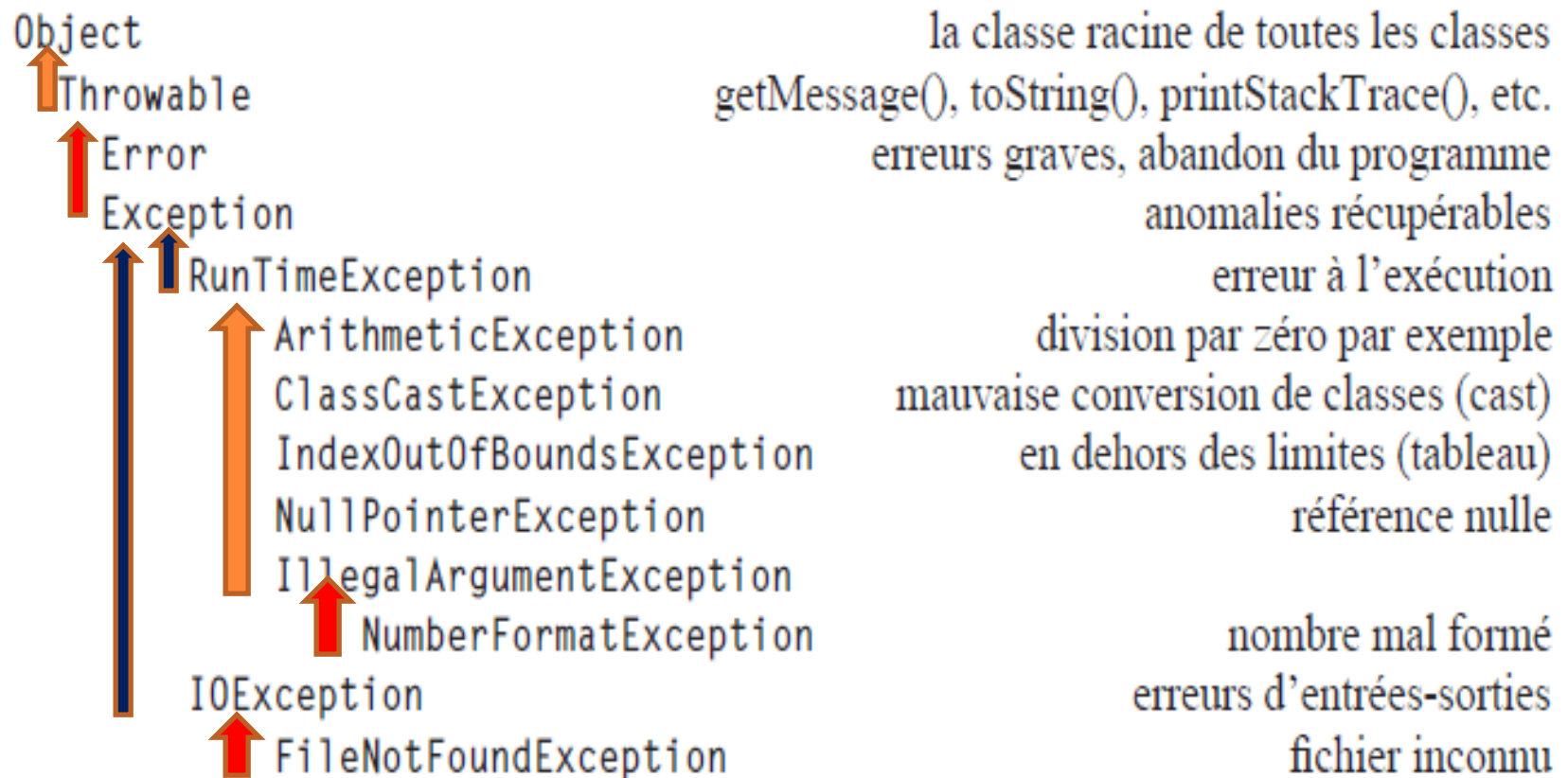
HIÉRARCHIE DES CLASSE D'EXCEPTION

- La sous-classe **Exception** représente des erreurs moins graves qui peuvent être gérées au moment de l'exécution. C'est la classe mère des classes **RuntimeException**, **IOException**, **SQLException**, etc.



HIÉRARCHIE DES CLASSE D'EXCEPTION

Les exceptions constituent un arbre hiérarchique utilisant la notion d'héritage.



GESTION DES EXCEPTIONS (BLOC TRY .. CATCH)

```
2 public class TestAvecException {
3 public static void main(String[] args) {
4     int i = 10;
5     int j = 0;
6
7     try {
8         System.out.println("résultat = " + (i / j));
9     } catch (ArithmeticException e) {
10         System.out.println("Erreur de division par zéro !");
11
12     }
13     System.out.println("En fin !!je pourrai exécuter maintenant!");
14 }
```

Problems Declaration Console

<terminated> TestAvecException [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\ja

Erreur de division par zéro !

En fin !!je pourrai exécuter maintenant!

GESTION DES EXCEPTIONS (BLOC TRY .. CATCH)

- Le bloc **catch** contient un objet de type **ArithmeticException** en paramètre. Nous l'avons appelé **e**.
- Comme l'exception de type **ArithmeticException** est capturée, le déroulement normal du programme est interrompu et l'instruction du bloc catch de même type s'exécute ! Par la suite, le programme principal reprend son déroulement normal.
- De ce fait, notre message d'erreur personnalisé s'affiche à l'écran.
- Maintenant, le paramètre de notre bloc catch (**e**), il sert à quoi, au juste ?

23

GESTION DES EXCEPTIONS (BLOC TRY .. CATCH)

- Il sert à savoir quel type d'exception doit être capturé.
- Voici le même test, en remplaçant l'instruction du catch par celle-ci :

```
try {  
    System.out.println("résultat = " + (i / j));  
} catch(ArithmeticException e) {  
    System.out.println("Division par zéro !" + e.getMessage());  
}  
  
//System.out.println("Erreur de division par zéro !");  
System.out.println("En fin !!je pourrai exécuter maintenant!");  
}
```

```
Division par zéro !/ by zero  
En fin !!je pourrai exécuter maintenant!
```

```

1 public class TestException {
2     public static void main(java.lang.String[] args) {
3         // Insert code to start the application here.
4         int i = 3;
5         int j = 0;
6         try {
7             System.out.println("résultat = " + (i / j));
8         } catch (ArithmeticException e) {
9             System.out.println("getMessage");
10            System.out.println(e.getMessage());
11            System.out.println(" ");
12            System.out.println("toString");
13            System.out.println(e.toString());
14            System.out.println(" ");
15            System.out.println("printStackTrace");
16            e.printStackTrace();
17        }
18    }

```

Résultat :

```

1 C:>java TestException
2 getMessage
3 / by zero
4
5 toString
6 java.lang.ArithmeticException: / by zero
7
8 printStackTrace
9 java.lang.ArithmeticException: / by zero
10      at tests.TestException.main(TestException.java:24)

```

PLUSIEURS EXCEPTIONS DANS UN BLOC TRY..CATCH..FINALLY

```
try {  
    operation_risquée1;  
    opération_risquée2;  
}  
catch (ExceptionDeClasseFilleException e) {traitements }  
catch (ExceptionParticulière e) { traitements }  
catch (ExceptionDeClasseMèreException e) { traitements }  
finally { traitement_pour_terminer_proprement; }
```

PLUSIEURS EXCEPTIONS DANS UN BLOC TRY..CATCH..FINALLY

- S'il y a **plusieurs types d'exceptions** à intercepter, il faut définir autant de blocs catch que de types d'événements.
- Il faut faire attention à l'ordre des clauses catch, un type d'exception d'une **sous-classe doit venir avant** un type d'une exception d'une **super-classe**.
- C'est dans le but de traiter **en premier** les exceptions les **plus précises (sous-classes)** avant les exceptions plus générales (super-classe), car le traitement des exceptions sera transmit au bloc de niveau supérieur (super-classe)

22

PLUSIEURS EXCEPTIONS DANS UN BLOC TRY..CATCH..FINALLY

- Si vous ne respectez pas l'ordre attendu, l'erreur de compilation suivante sera produite. **Exemple:**

```
public class TestException {  
    public static void main(java.lang.String[] args) {  
        int i = 3;  int j = 0;  
        try { System.out.println("résultat = " + (i / j));  
        } catch (Exception e) {}  
        catch (ArithmeticException e) {}  
    }  
}
```

```
TestException.java:11: catch not reached.  
        catch (ArithmeticException e) {  
            ^  
1 error
```

PLUSIEURS EXCEPTIONS DANS UN BLOC TRY..CATCH..FINALLY

- le message précise que l'exception **ArithmeticException** est déjà attrapée.
- Effectivement, comme la classe `java.lang.ArithmeticException` dérive de la classe `java.lang.Exception`, elle est déjà gérée par le premier bloc catch associé au type **Exception**.

PLUSIEURS EXCEPTIONS DANS UN BLOC TRY..CATCH..FINALLY

- Le bloc **finally** est en général utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).
- Dans tout les cas de l'exécution ou non du catch, le bloc **finally** est exécutée mais il est optionnel.

LES EXCEPTIONS PRÉDÉFINIES

- Toute méthode pouvant lancer une exception contrôlée doit contenir soit un bloc try/catch/finally (si elle traite l'exception localement).
- Soit utilise le mot clé **throws** (si l'exception est traitée par une autre méthode).

23

LES EXCEPTIONS PRÉDÉFINIES

- le mot clé **throws** signifie qu'une **méthode est susceptible de lever une exception** mais elle ne peut pas la traiter localement.

```
public void ma_methode (int x) throws IOException {  
    ...  
}
```

→ Il est également possible de désigner l'interception de plusieurs exceptions :

```
public void ma_methode (int x) throws IOException, EOFException{  
    ...  
}
```



DÉFINIR UNE CLASSE D'EXCEPTION

- Jusqu'à présent on a parlé des exceptions prédéfinies qui se déclenchent toutes seules. Java offre au programmeur la possibilité de définir ses propres exceptions. Ces exceptions doivent hériter de la classe `Exception`. Le programmeur doit lui-même lever ses exceptions.
- Pour se faire Java met à sa disposition le mot-clé **throw** (à ne pas confondre avec **throws**).

DÉFINIR UNE CLASSE D'EXCEPTION

throws:

- Ce mot clé permet de dire à une instruction Java (condition, déclaration de variable...) ou à une classe entière qu'une exception potentielle sera gérée par une autre classe.
- Ce mot clé **est suivi** du nom de la classe qui va gérer l'exception. Ceci a pour but de définir le type d'exception qui risque d'être générée par l'instruction, ou la classe qui précède le mot clé **throws**.

DÉFINIR UNE CLASSE D'EXCEPTION

throw:

- Celui-ci permet d'instancier un objet dans la classe suivant l'instruction **throws**. Cette instruction est suivie du mot clé **new** ainsi que d'un objet cité avec **throws**. En fait, il lance une exception, tout simplement.

23

UTILISER UNE CLASSE D'EXCEPTION DÉFINIE PAR LE PROGRAMMEUR

```
1 package ExecpPersonalisé;
2
3 public class Ville{
4     |
5     String nomVille;
6     int nbreHabitant;
7     public Ville(String nomVille, int nbreHabitant){
8         this.nomVille = nomVille;
9         this.nbreHabitant = nbreHabitant;
10    }
11 }
```

Q: Si l'utilisateur crée une instance ville avec un nombre d'habitants négatif ?

R: pour remédier, le programmeur doit créer un objet de type NombreHabitantException (qui hérite d'Exception).

Exemple: NombreHabitantException extends Exception

UTILISER UNE CLASSE D'EXCEPTION DÉFINIE PAR LE PROGRAMMEUR

```
1 package ExecpPersonalisé;
2
3 public class NombreHabitantException extends Exception{
4
5     public NombreHabitantException(){
6         |
7         System.out.print("Vous essayez d'instancier une classe Ville");
8         System.out.println("avec un nombre d'habitants négatif !");
9     }
10
11 }
```


CAPTURER UNE EXCEPTION DÉFINIE PAR LE PROGRAMMEUR

```
1 package ExecpPersonalisé;
2
3 public class Ville{
4
5     String nomVille;
6     int nbreHabitant;
7     public Ville(String nVil, int nHab) throws NombreHabitantException
8     {
9         if(nHab < 0)
10             throw new NombreHabitantException();
11         else
12         {
13             nomVille = nVil;
14             nbreHabitant = nHab;
15         }
16     }
17 }
```

CAPTURER UNE EXCEPTION DÉFINIE PAR LE PROGRAMMEUR

- **throws NombreHabitantException** nous indique que si une erreur est capturée, celle-ci sera traitée en tant qu'objet de la classe **NombreHabitantException** ! Ce qui, au final, nous renseigne sur le type de l'erreur en question.

throw new NombreHabitantException();
instancie la classe **NombreHabitantException** si la condition **if(nHab < 0)** est remplie.

Merci ...

SUITE ...

- Maintenant, pour créer un objet ville, le constructeur est devenu une **méthode à risque**, vous devrez gérer les exceptions possibles sur cette instruction. Avec un bloc **try{} catch{}.**

```
3 public class Main {  
4     public static void main(String[] args){  
5  
6         Ville v = new Ville("Blida", -15000);  
7     }  
8 }  
9
```

SUITE ...

```
3 public class Main {  
4     public static void main(String[] args){  
5         try {  
6             Ville v = new Ville("Blida", -15000);  
7         } catch (NombreHabitantException e) {System.out.println("e = "+ e);}  
8     }  
9 }
```

Problems Declaration Console

<terminated> Main (6) [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (15 déc. 2021 à 11:50:43)

Vous essayez d'instancier une classe Ville avec un nombre d'habitants négatif !
e = [ExecpPersonalisé.NombreHabitantException](#)

- Maintenant ,l'erreur a disparu et le code compile et s'exécute correctement.

LA GESTION DE PLUSIEURS EXCEPTIONS

- nous voulons lever une deuxième exception, si le nom de la ville fait moins de 3 caractères.

```
3 public class NomVilleException extends Exception {  
4     public NomVilleException(String message){  
5         super(message);  
6     }  
7 }
```

Avec cette redéfinition (super), nous pourrions afficher notre message d'erreur en utilisant la méthode **getMessage()**.

LA GESTION DE PLUSIEURS EXCEPTIONS

```
public class Ville{  
    String nomVille;  
    int nbreHabitant;  
    public Ville(String nVil, int nHab) throws NombreHabitantException, NomVilleException  
    {  
        if(nHab < 0) throw new NombreHabitantException();  
        if(nVil.length() < 3)  
            throw new NomVilleException("le nom de la ville est inférieur à 3 caractères ! nom  
        else  
        {  
            nomVille = nVil;  
            nbreHabitant = nHab;  
        }  
    }  
}
```


LA GESTION DE PLUSIEURS EXCEPTIONS

```
public static void main(String[] args) {  
    | try {  
        Ville v = new Ville("Bl", -15000);  
    }  
    //Gestion de l'exception sur le nombre d'habitants  
    catch (NombreHabitantException e) {  
    }  
    //Gestion de l'exception sur le nom de la ville  
    catch(NomVilleException e2){  
        System.out.println(e2.getMessage()); }  
    System.out.println(v.toString());  
    }  
}
```

Vous pouvez constater qu'Eclipse affiche une erreur !!

R: car la déclaration de l'objet **Ville** est faite dans le premier sous-bloc **try{}** et une variable déclarée dans un bloc d'instructions n'existe que dans ce bloc d'instructions ! **Pour pallier ce problème**, il nous suffit de déclarer notre objet en dehors du bloc **try{}** et de l'instancier à l'intérieur !

LA GESTION DE PLUSIEURS EXCEPTIONS.

```
8      Ville v = null;
9      try {
10         v = new Ville("B1", -15000);
11     }
12     //Gestion de l'exception sur le nombre d'habitants
13     catch (NombreHabitantException e) { }
14     //Gestion de l'exception sur le nom de la ville
15     catch (NomVilleException e2){
16         System.out.println(e2.getMessage()); }
17     System.out.println(v.toString());
18 }
```

Problems Declaration Console

<terminated> Main (6) [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (15 déc. 2021 à 12:37:28)

Vous essayez d'instancier une classe Ville avec un nombre d'habitants négatif
Exception in thread "main" [java.lang.NullPointerException](#)
at ExcepPersonalisé.Main.main([Main.java:19](#))

lorsque nous arrivons sur l'instruction
"System.out.println(v.toString());", notre objet est **null** !
Donc, Une **NullPointerException** est levée !

LA GESTION DE PLUSIEURS EXCEPTIONS

```
8      Ville v = null;
9      try {
10         v = new Ville("Bl", -15000);
11     }
12     //Gestion de l'exception sur le nombre d'habitants
13     catch (NombreHabitantException e) { }
14     //Gestion de l'exception sur le nom de la ville
15     catch(NomVilleException e2){
16         System.out.println(e2.getMessage()); }
17     System.out.println(v.toString());
18 } }
```

Problems Declaration Console

<terminated> Main (6) [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (15 déc. 2021 à 12:37:28)

Vous essayez d'instancier une classe Ville avec un nombre d'habitants négatif
Exception in thread "main" [java.lang.NullPointerException](#)
at ExcepPersonalisé.Main.main(Main.java:19)

Pour **pallier ce problème**, Il suffit d'instancier un objet Ville par défaut dans le bloc **catch{}**. Grâce à cela, si notre instanciation avec valeur échoue, on fait une instanciation par défaut qui n'est pas une méthode à risque !.,

LA GESTION DE PLUSIEURS EXCEPTIONS

```
8     Ville v = null;
9     try {
10         v = new Ville("B1", -15000);
11     }
12     //Gestion de l'exception sur le nombre d'habitants
13     catch (NombreHabitantException e) { v = new Ville(); }
14     //Gestion de l'exception sur le nom de la ville
15     catch (NomVilleException e2){
16         System.out.println(e2.getMessage());
17         v = new Ville();}
18     System.out.println(v.toString());
19 }
```

Problems Declaration Console

<terminated> Main (6) [Java Application] C:\Program Files\Java\jre1.8.0_281\bin\javaw.exe (15 déc. 2021 à 15:50:12)

Vous essayez d'instancier une classe Ville avec un nombre d'habitants négatif
ExecpPersonalisé.Ville@15db9742

Si vous mettez un nom de ville de moins de 3 caractères, et un nombre d'habitants négatif, c'est l'exception du nombre d'habitants qui sera levée en premier ! Et pour cause... il s'agit de notre première condition dans notre constructeur

Merci !!