Gestion des exceptions

- **Introduction**
- **Exceptions**
- **Types d'exceptions**
- **Traitement d'une exception**
- **Lancement d'une exception**

DR. YOUSSEF ELMIR ELMIR@ESTIN.DZ

INTRODUCTION

- Introduction

  - Un logiciel est **correct** s'il accomplit bien la tâche pour laquelle il a été conçu.

  - Un logiciel est **robuste** s'il est capable de fonctionner même en présence d'erreurs.

Exemple: Un programme qui lit des nombres entrés par l'utilisateur et les affiche dans l'ordre croissant est correct s'il affiche le bon résultat quelques soient les nombres entrés. Il est robuste si, en plus, il traite les données non numériques en affichant un message d'erreur ou en les ignorant .

- Introduction

Durant le développement et l'exécution d'une application, plusieurs erreurs peuvent se produire:

- Les erreurs de syntaxe: très fréquentes, elles se produisent au moment de la compilation (ex: mot clé mal orthographie).
- Les erreurs d'exécution: apparaissent pendant l'exécution. Syntaxe correcte mais l'environnement de l'application ne permet pas d'exécuter une instruction du programme (ouverture de fichier, erreurs de pointeurs, dépassement de capacité…)
- Les erreurs de logique: Aucune erreur de syntaxe ni d'exécution mais les résultats obtenus ne sont pas ceux attendus

- Introduction

  - Même si un programme est correct, son exécution peut être interrompue brutalement à cause d'évènements exceptionnels (types de données incorrects, fin prématurée de tableaux ou de fichiers..etc). Ces évènements sont appelés Exceptions.

« Une exception est un évènement qui arrive durant l'exécution du programme et qui perturbe le déroulement normal de celui-ci »

- Introduction

  - Le concepteur d'un programme doit essayer dans un premier temps de prévoir (anticiper) les erreurs possibles soit:
    - En ajoutant des séquences de tests pour les empêcher,
    - En mettant en place un système pour signaler un fonctionnement anormal du programme en retournant des valeurs spécifiques

  - Mais aucune de ces solutions n'est satisfaisante car:
    - il est très difficile de dénombrer toutes les erreurs possibles sans en omettre aucune,
    - l'écriture du code devient fastidieuse et complexe et sa maintenance difficile à cause de son illisibilité (noyé dans des imbrications de tests)

Gestion des exceptions

- Exceptions
  - Java dispose d'un mécanisme très souple nommé gestion des exceptions qui permet de détecter et traiter les erreurs pouvant surgir lors de l'exécution d'un programme.
  - Ce mécanisme est utilisé pour traiter le fonctionnement anormal d'une partie du code.

- Exceptions
  - Exemple

```
public class TestException {
public static void main(java.lang.String[] args) {
int i = Integer.parseInt (args [0]);
int j = Integer.parseInt (args [1]);
System.out.println("résultat = " + (i / j));
} }
```

  - **Si l'utilisateur saisit un zéro comme deuxième valeur il verra s'afficher le message suivant:**

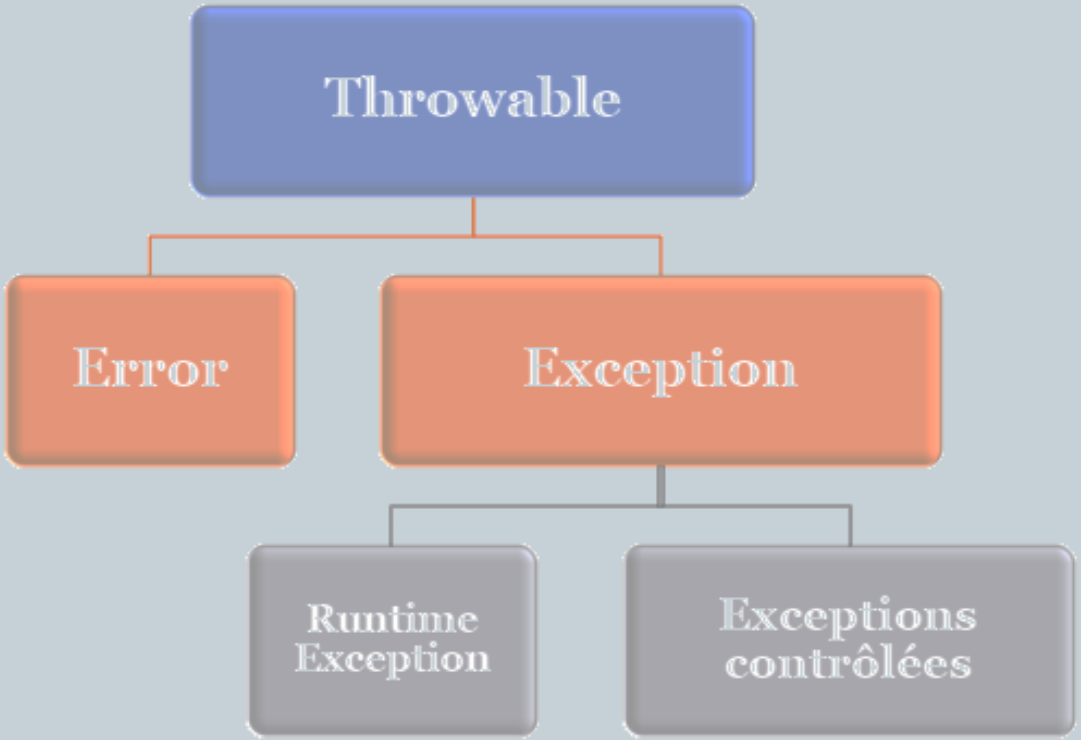
```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
TestException.main(TestException.java:6)
```

TYPES D'EXCEPTIONS

Gestion des exceptions

- Types d'exceptions
  - Il existe trois types d'exceptions:
    - Error (erreurs graves) ;
    - Exception (erreurs communes) ;
      - RuntimeException (erreurs langage) ;

- Types d'exceptions
  - Arbre d'héritage des exceptions



Gestion des exceptions

- Types d'exceptions
  - Types :
    1. Les exceptions de type Error
       - Ce sont des évènements exceptionnels externes au programme (problèmes de chargement des classes en mémoire, problème matériel…) et qui ne peuvent pas être anticipés ou récupérés.
       - Ils sont de type Error ou ses sous-classes (Virtual Machine Error, IOError, etc.)

Exemple: Supposons qu'une application puisse ouvrir un fichier correctement mais ne puisse pas le lire à cause d'une erreur matérielle ou du système d'exploitation. L'incapacité de lire le fichier lancera une exception de type java.io.IOError

Gestion des exceptions

- Types d'exceptions
  - Types :
    2. Exceptions contrôlées
       - Ce sont les exceptions qu'un programme bien écrit doit anticiper et récupérer (dit aussi attraper, saisir ou traiter). Ce sont des exceptions de type Exception

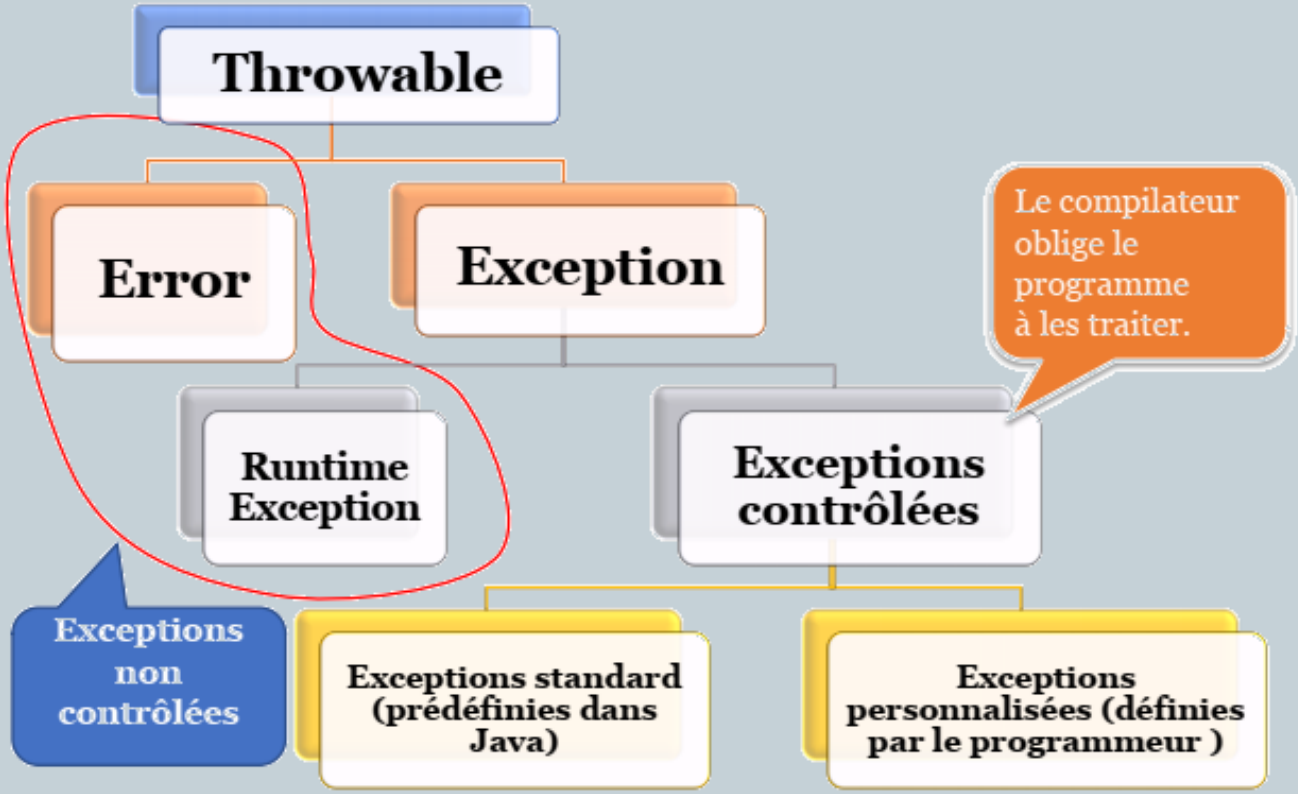
Exemple : Une application invite l'utilisateur à entrer le nom d'un fichier. L'utilisateur peut donner le nom d'un fichier qui n'existe pas. Dans ce cas une exception du type java.io.FileNotFound sera lancée. Un programme bien écrit devrait prévenir cette exception en signalant à l'utilisateur que le fichier n'existe pas et en l'invitant à taper le nom d'un fichier existant.

- Types d'exceptions
  - Types :
    3. Les exceptions de type RuntimeException
       - RuntimeException est une sous classe de Exception
       - Ce sont des évènements exceptionnels internes à la JVM et au programme (liées au langage) et qui peuvent être traités par le programmeur.
       - Ces exceptions comprennent les exceptions arithmétiques (par exemple la division par zéro), les exceptions de pointeur (lors d'une tentative d'accès à un objet par l'intermédiaire d'une référence null), et les exceptions d'indexation (valeur d'index hors bornes).

       Exemple: Si dans une application, au lieu de passer le nom d'un fichier à une méthode, on passe null, alors une exception du type NullPointerException sera lancée

Gestion des exceptions

- Types d'exceptions
  - Arbre d'héritage des exceptions



TRAITEMENT D'UNE EXCEPTION

Gestion des exceptions

- Traitement d'une exception

  - Une exception contrôlée est une exception qui hérite de la classe Exception. Elle peut être standard (définie dans JDK) ou bien personnalisée.

  - Un programme java bien écrit qui contient une partie du code pouvant générer (ou provoquer) une exception contrôlée doit essayer de la traiter (ou récupérer) de l'une des deux manières suivantes

    1. En la traitant avec le bloc try\catch
    2. En la propageant avec le mot clé throws

- Traitement d'une exception
  - Le bloc try-catch(1)

    - La clause "try" s'applique à un bloc d'instructions correspondant au fonctionnement normal mais pouvant générer des erreurs.

```
try {
.....
}
```

Gestion des exceptions

- Traitement d'une exception
  - Le bloc try-catch(2)

    - La clause catch s'applique à un bloc d'instructions définissant le traitement d'un type d'erreur. Ce traitement sera lancé sur une instance de la classe d'exception passée en paramètre.

```
try {
... }
catch (ExceptionType1 e1) {
... }
catch(ExceptionType2 e2) {
... }
```

Gestion des exceptions

- Traitement d'une exception
  - Le bloc try-catch(3)

Exemple :

```
public class TestException2 {
public static void main(java.lang.String[] args) {
// Insert code to start the application here.
int i = Integer.parseInt (args [0]);
int j = Integer.parseInt (args [1]);
try {
System.out.println("résultat = " + (i / j));
}
catch (ArithmeticException e) {
System.out.println("Division par zéro ");
} } }
```

- Traitement d'une exception
  - Le bloc try-catch(4)

Que se passe t-il quand une exception est levée ?

- A tout moment, une instruction ou une méthode peut lever(lancer) une exception
- Une méthode peut attraper (récupérer, saisir) une exception ou bien la laisser se propager en la laissant remonter vers la méthode appelante qui peut elle-même l'attraper ou la laisser remonter

- Traitement d'une exception
  - Le bloc try-catch(5)

Que se passe t-il quand une exception est levée ?

- **Cas 1. En dehors d'un bloc try**

1. La méthode retourne immédiatement, l'exception remonte vers la méthode appelante
2. La main est donnée à la méthode appelante
3. L'exception peut alors éventuellement être attrapée et traitée par cette méthode appelante ou remonter dans la pile

- Traitement d'une exception
  - Le bloc try-catch(6)

Que se passe t-il quand une exception est levée ?

- **Cas 2. Dans un bloc try**
  - Si une des instructions du bloc try provoque une exception
  1. Les instructions suivantes du bloc try ne sont pas exécutées
  2. Si au moins une des clauses catch correspond au type de l'exception,
     - la première clause catch appropriée est exécutée
     - l'exécution se poursuit juste après le bloc try/catch

     Sinon,
     - la méthode retourne immédiatement
     - l'exception remonte vers la méthode appelante

- Traitement d'une exception
  - Le bloc try-catch(7)

Que se passe t-il quand une exception est levée ?

1. **Dans les cas où l'exécution des instructions de la clause try ne provoque pas d'exceptions**
   - **le déroulement du bloc de la clause try se déroule comme s'il n'y avait pas de bloc try-catch**
   - **le programme se poursuit après le bloc try-catch**

2. **Si le bloc catch est vide (aucune instruction entre les accolades) alors l'exception capturée est ignorée.**

Une telle utilisation de l'instruction try/catch n'est pas une bonne pratique : il est préférable de toujours apporter un traitement adapté lors de la capture d'une exception.

- Traitement d'une exception
  - Le bloc try-catch(8)

Que se passe t-il quand une exception est levée ?

3. **Eviter d'attraper une exception en mettant dans le bloc catch un simple affichage de message qui ne donne pas d'informations sur le problème. Au minimum afficher ce qu'affiche la méthode printStackTrace afin de pouvoir résoudre le problème**

4. **Si une variable est déclarée à l'intérieur d'un bloc try, elle ne peut pas être utilisée à l'intérieur d'un bloc catch. Il faut déclarer les variables communes à l'extérieur du bloc try/catch**

- Traitement d'une exception
  - Le bloc try-catch(9)

Et si l'exception n'est pas traitée?

Si une exception remonte jusqu'à la méthode main sans être traitée par cette méthode (donc pas traitée par les autres méthodes non plus),

- L'exécution du programme est stoppée
- Le message associé à l'exception est affiché, avec une description de la pile des méthodes traversées par l'exception

- Traitement d'une exception
  - Gestion de plusieurs exceptions

    - S'il y a plusieurs types d'erreurs et d'exceptions à intercepter, il faut définir autant de bloc catch que de types d'événements.

    - Par type d'exception, il faut comprendre « qui est du type de la classe de l'exception ou d'une de ses sous classes ». Ainsi dans l'ordre séquentiel des clauses catch, **un type d'exception ne doit pas venir après un type d'une exception d'une super classe**.

    - Il faut faire attention à **l'ordre** des clauses **catch** pour traiter en premier les exceptions les plus précises (sous classes) avant les exceptions les plus générales. Un message d'erreur est émis par le compilateur dans le cas contraire.

Gestion des exceptions

- Traitement d'une exception
  - Gestion de plusieurs exceptions

```
public class TestException2 {
public static void main(String[] args) {
// Insert code to start the application here.
int i = Integer.parseInt (args [0]);
int j = Integer.parseInt (args [1]);
try {
System.out.println("résultat = " + (i / j));
}
catch (Exception e) {
}
catch (ArithmeticException e) {
System.out.println("Divion par zéro ");
} } }
```

Bloc catch inaccessible

Gestion des exceptions

- Traitement d'une exception
  - Le bloc Finally

    - Nous avons vu qu'une exception provoque l'arrêt de l'exécution d'une méthode pour se brancher vers le bloc catch adéquat.

    - Mais on peut introduire un bloc d'instructions qui seront toujours exécutées (qu'une exception soit levée ou pas).
      - Soit après le bloc try si celui-ci s'est bien déroulé(aucune exception n'est déclenchée)
      - Soit après le bloc catch dans le cas où une exception a été déclenchée

Gestion des exceptions

- Traitement d'une exception
  - Le bloc Finally

    - Ce bloc est introduit par le mot-clé finally et doit obligatoirement être placé après le dernier bloc catch.

```
try { ..... }
catch (Ex e) { ..... }
finally
{ // instructions
}
```

    - Remarques: Les instructions dans le bloc finally sont souvent utilisées pour faire des actions de « nettoyage » (ou de libération de ressources), comme fermer un fichier ou une connexion réseau.

Gestion des exceptions

- Traitement d'une exception
  - Le bloc Finally

    - Ce bloc est introduit par le mot-clé finally et doit obligatoirement être placé après le dernier bloc catch.

```
try { ..... }
catch (Ex e) { ..... }
finally
{ // instructions
}
```

    - Remarques: Les instructions dans le bloc finally sont souvent utilisées pour faire des actions de « nettoyage » (ou de libération de ressources), comme fermer un fichier ou une connexion réseau.

LANCEMENT D'UNE EXCEPTION

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Le mot clé throws
      - Toute méthode pouvant lancer une exception contrôlée doit contenir soit un bloc try/catch (si elle traite l'exception localement) ou bien utiliser le mot clé throws (si l'exception est propagée: traitée par une autre méthode)
      - Nous avons vu comment utiliser le bloc try/ catch, maintenant nous allons voir l'utilisation de throws

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Le mot clé throws
      - Exemple

```
public class TestThrows {
public void test(String[] a)throws
ArithmeticException
{
int i = Integer.parseInt (a[0]);
int j = Integer.parseInt (a[1]);
System.out.println("résultat = " + (i / j));
}
…
}
```

Si une Exception se produit ici elle ne sera pas traitée localement, elle va remonter comme une « bulle » dans la pile d'exécution pour trouver une portion de code apte à la traiter

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Le mot clé throws
      - Le programmeur peut lancer lui-même des exceptions depuis les méthodes qu'il écrit
        - Il peut lancer des exceptions des classes d'exceptions fournies par le JDK
        - Ou lancer des exceptions appartenant à de nouvelles classes d'exceptions, qu'il a lui-même écrites, et qui sont mieux adaptées aux classes qu'il a écrites.
      - Si on lance une exception dans une méthode avec throw, il faut l'indiquer dans la déclaration de la méthode, en utilisant le mot clé throws.

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Le mot clé throws
      - exemple

```
public class TestThrow {
public void test(String[] a) throws
ArithmeticException{
int i = Integer.parseInt (a [0]);
int j = Integer.parseInt (a [1]);
if(j==0) throw new ArithmeticException();
System.out.println("résultat = " + (i / j));
}
}
```

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Les exceptions personnalisées
      - En cas de nécessité, on peut créer ses propres exceptions en créant des sous classes de la classe Exception (par convention, le mot « Exception » est inclus dans le nom de la nouvelle classe).
      - Il suffit ensuite d'utiliser le mot clé throw, suivi d'un objet dont la classe dérive de la classe créée.

- Lancement d'une exception
  - L'utilisation de throws
    - Les exceptions personnalisées
      - Exemple
        - Supposons que l 'on souhaite manipuler des points ayant des coordonnées non négatives. Nous utiliserons donc une classe Point ayant un constructeur à deux arguments.
        - Au sein de ce constructeur, on vérifiera la validité des coordonnées entrées par l'utilisateur, si l'une d'entre elles est incorrecte, nous déclencherons une exception à l'aide du mot clé throw.

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Les exceptions personnalisées
      - Exemple
        - L'instruction throw a besoin d'avoir un objet du type de l'exception concernée
        - Nous créerons donc une classe que nous appellerons CoordonneesException qui dérive de Exception

```
class CoordonneesException extends Exception {
}
```

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Les exceptions personnalisées
      - Exemple

```
class Point
{
private int x;
private int y;
public Point(int x, int y) throws CoordonneesException
{
if((x<0) || (y<0)) throw new CoordonneesException();
this.x = x;
this.y = y;
}}
```

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Les exceptions personnalisées
      - Exemple

```
public class ExceptionPersonnalisee {
public static void main(String args[]){
int x = Integer.parseInt(args[0]);
int y = Integer.parseInt (args[1]);
try{
Point p = new Point (x,y);
}
catch(CoordonneesException e)
{
System.out.println(" coordonnée négative ");
} } }
```

Gestion des exceptions

- Lancement d'une exception
  - L'utilisation de throws
    - Remarque
    - Si une méthode m2 avec un en-tête qui contient « throws XXException » et une méthode m1 fait un appel à m2, alors
      - ▫ soit m1 insère l'appel de m2 dans un bloc « try - catch(XXException) »
      - soit m1 déclare (en utilisant le mot clé throws) qu'elle peut lancer une XXException (ou un type plus large)

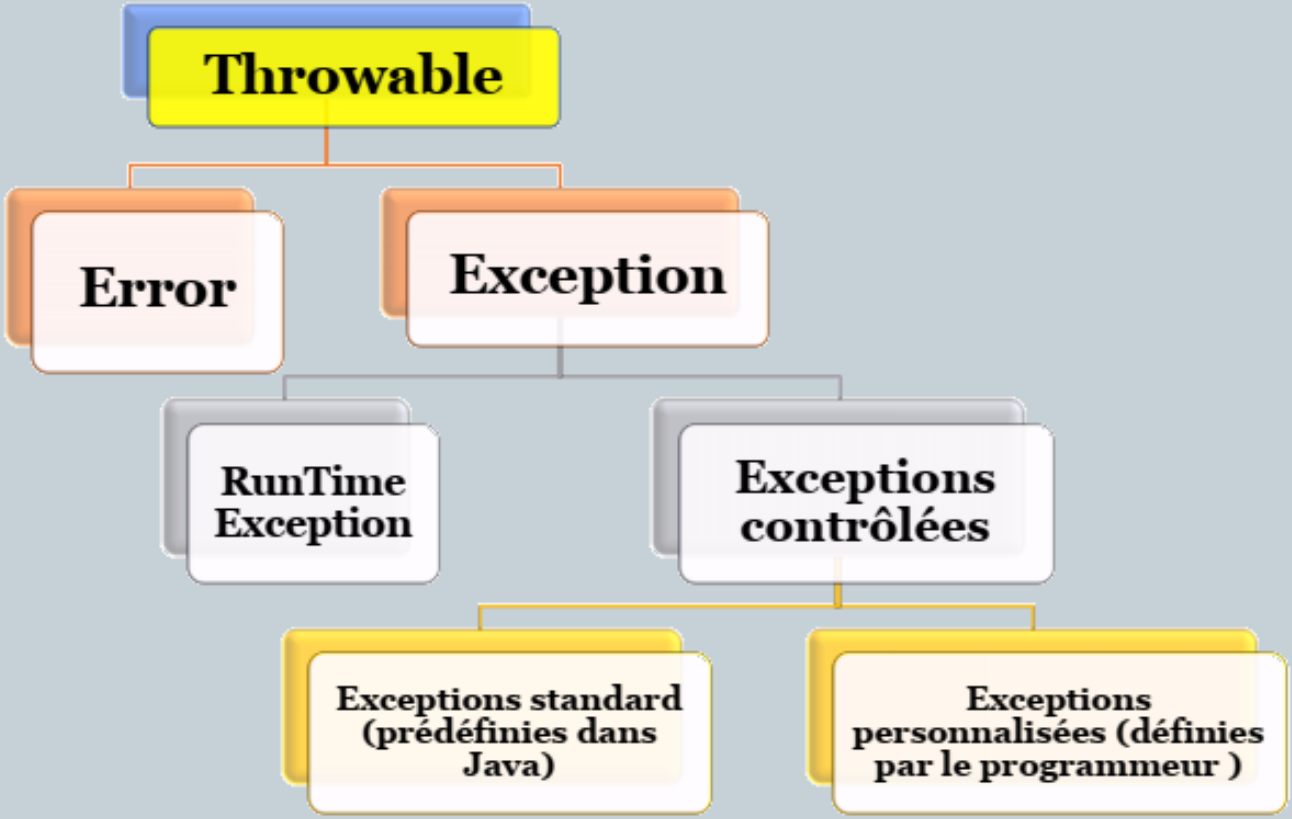
- Lancement d'une exception
  - La classe Throwable

    - C'est la classe mère de toutes les exceptions et elle descend directement de la classe Object.

    - Ses principales méthodes sont :
      - String getMessage( ) retourne le message
      - void printStackTrace( ) affiche le message d'erreur et la pile des appels de méthodes qui ont conduit au problème
      - void printStackTrace(PrintStream s) Idem mais envoie le résultat dans un flux

Gestion des exceptions

- Lancement d'une exception
  - **Arbre d'héritage des exceptions**



Gestion des exceptions

- Lancement d'une exception
  - **La classe Throwable**

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

Gestion des exceptions

- Lancement d'une exception
  - **La classe Throwable**
    - **Résultat de l'exécution**

getmessage

/ by zero

printStackTrace

java.lang.ArithmeticException: / by zero at TestMethodesThrowable.main(TestMethodesThrowable.java:6)