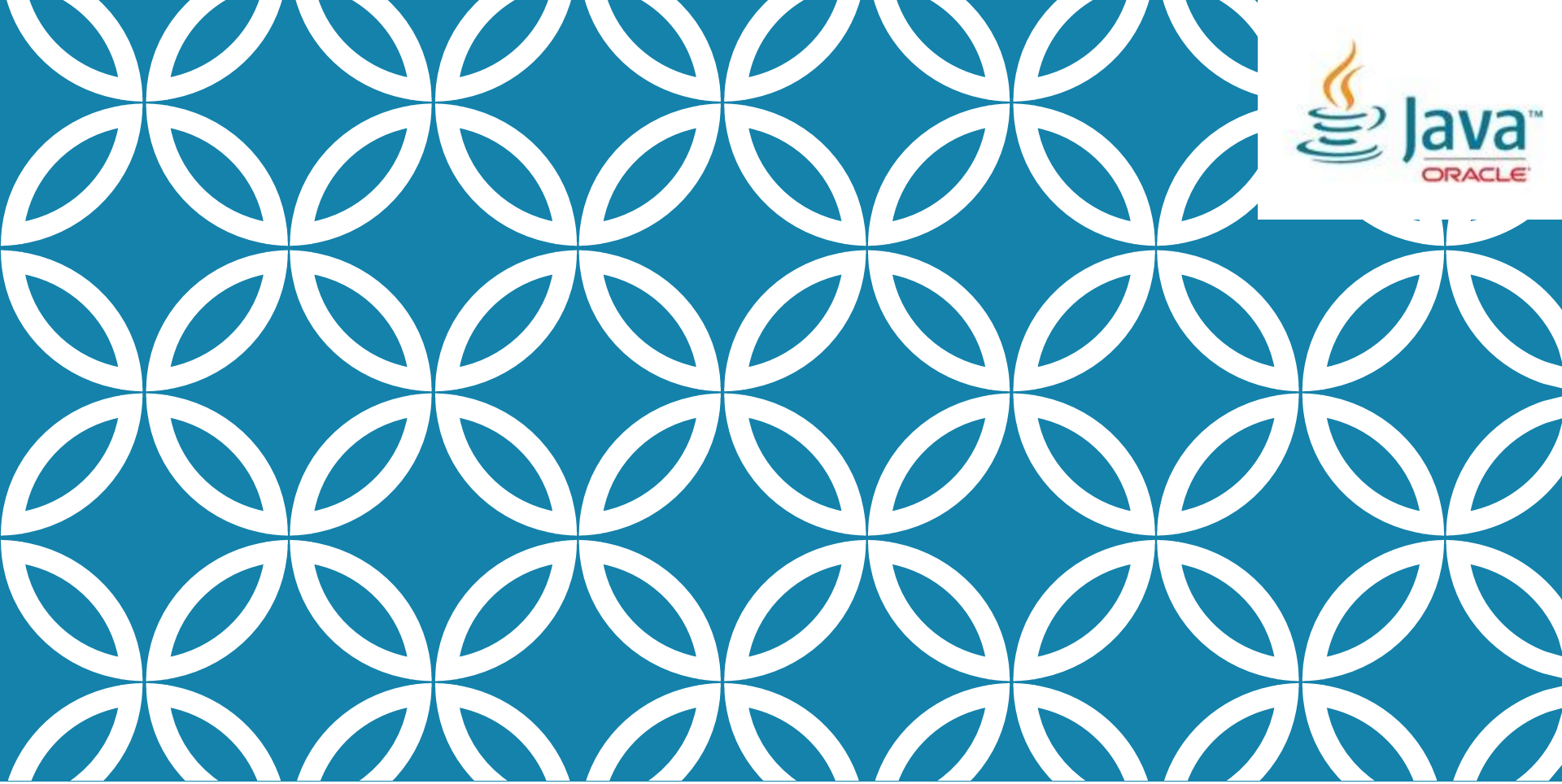


COURS 8
LES EXCEPTIONS EN JAVA

Par Aïcha El Golli

aicha.elgolli@essai.ucar.tn

ECOLE SUPÉRIEURE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

DÉFINITIONS

Une exception est chargée de signaler un comportement exceptionnel (mais prévu) d'une partie spécifique d'un logiciel. Dans les langages de programmation actuels, les exceptions font partie du langage lui-même. C'est le cas de Java qui intègre les exceptions comme une classe particulière : la classe Exception. Cette classe contient un nombre important de classes dérivées.

Programme sans gestion de l'exception

Soit un programme Java contenant un incident d'exécution (une division par zéro dans l'instruction x = y/z;) dans la méthode meth() de la classe Action, cette méthode est appelée dans la classe Cours à travers un objet de classe Action :

```
class Action{
public void meth(){
int x, y=5, z=0;
System.out.println("...avant erreur");
x=y/z;
System.out.println("...après erreur");
} }
public class Cours {
public static void main(String[] args) {
Action a=new Action();
System.out.println("début programme.");
a.meth();
System.out.println("fin programme.");
}}
```

EXEMPLE SANS GESTION D'EXCEPTIONS

Lors de l'exécution, après avoir affiché les chaînes « **début du programme.**" et " **...avant erreur**", le programme s'arrête et la java machine signale une erreur. Voici ci-dessous l'affichage obtenu sur la console lors de l'exécution :

---- java Cours
début du programme.
...avant erreur
java.lang.ArithmeticException : / by zero
---- : Exception in thread "main"

La méthode main :
- a instancié un objet a de classe Action;
- a affiché sur la console la phrase « début du programme.",
- a invoqué la méthode meth() de l'objet a,
- a affiché sur la console la phrase " ...avant erreur",
- a exécuté l'instruction "x = y/z;"

Dès que l'instruction "x = y/z;" a été exécutée celle-ci a provoqué un incident. **En fait une exception de la classe ArithmeticException a été "levée" (un objet de cette classe a été instancié) par la Java machine.**

La Java machine a arrêté le programme immédiatement à cet endroit parce qu'elle n'a pas trouvé de code d'interception de cette exception qui a été automatiquement levée.
Nous allons voir comment intercepter (on dit aussi "attraper" - to catch) cette exception afin de faire réagir notre programme afin qu'il ne s'arrête pas brutalement.

DÉFINITION

Une exception est un signal

- qui indique que quelque chose d'exceptionnel (par exemple une erreur) s'est produit,
- qui interrompt le flot d'exécution normal du programme.

lancer (throw) une exception consiste à signaler ce quelque chose,

attraper (catch) une exception permet d'exécuter les actions nécessaires pour traiter cette situation.

si une exception n'est jamais attrapée :

- propagation jusqu'à la méthode main() à partir de laquelle l'exécution du programme a débuté,
- affichage d'un message d'erreur et de la trace de la pile des appels (call stack),
- arrêt de l'exécution du programme.

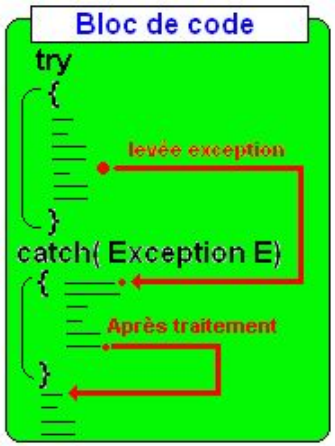
Exemple avec gestion d'exceptions

Java possède une instruction de gestion des exceptions, qui permet d'intercepter des exceptions dérivant de la classe Exception :
try ... catch

Syntaxe minimale d'un tel gestionnaire :

```
try {
        <lignes de code à protéger>
}
catch ( UneException E ) {
<lignes de code réagissant à l'exceptionUneException >
}
```

Le type **UneException** est obligatoirement une classe qui **hérite de la classe Exception.**
Schéma du fonctionnement d'un tel gestionnaire :



Le gestionnaire d'exception "déroute" l'exécution du programme vers le bloc d'interception catch qui traite l'exception (exécute le code contenu dans le bloc catch), puis renvoie et continue l'exécution du programme vers le code situé après le gestionnaire lui-même.

PRINCIPE DE FONCTIONNEMENT DE L'INTERCEPTION

Dès qu'une **exception est levée** (instanciée), la Java machine **stoppe immédiatement** l'exécution normale du programme à la **recherche d'un gestionnaire** d'exception susceptible d'intercepter (saisir) et traiter cette exception. Cette recherche s'effectue à partir du **bloc englobant** et se poursuit sur les blocs plus englobant si aucun gestionnaire de **cette exception** n'a été trouvé.

```
class Action{
public void meth(){
int x, y=5, z=0;
System.out.println("...avant erreur");
x=y/z;
System.out.println("...après erreur");
}
}
```

Engendre une exception

```
public class Cours {
public static void main(String[] args) {
Action a=new Action();
System.out.println("début programme.");
try { a.meth(); }
catch (ArithmeticException E) {
System.out.println("Interception exception");
}
System.out.println("fin programme."); }}
```

Levée d'ArithmeticException

Traitement, puis poursuite de l'exécution

l'affichage obtenu sur la console lors de l'exécution de ce programme :

----javaCours

début programme.

...avant erreur

Interception exception

fin programme.

---- : operation complete.

Nous remarquons donc que la Java machine a donc bien exécuté le code d'interception situé dans le corps du "catch(ArithmeticException E){...}" et a poursuivi l'exécution normale après le gestionnaire.

Le gestionnaire d'exception se situe dans la méthode main (code englobant) qui appelle la méthode meth() qui lève l'exception.

INTERCEPTIONS DE PLUSIEURS EXCEPTIONS

Dans un gestionnaire try...catch, il est en fait possible d'intercepter plusieurs types d'exceptions différentes et de les traiter.

Nous montrons la syntaxe d'un tel gestionnaire qui fonctionne comme un sélecteur ordonné, ce qui signifie qu'**une seule clause d'interception est exécutée.**

Dès qu'une exception intervient dans le < bloc de code à protéger>, la Java machine scrute séquentiellement toutes les clauses catch de la première jusqu'à la nième. Si l'exception actuellement levée est d'un des types présents dans la liste des clauses le traitement associé est effectué, la scrutation est abandonnée et le programme poursuit son exécution après le gestionnaire.

Syntaxe :

```
try {
   < bloc de code à protéger>
  }
  catch ( TypeException1 E ) { <Traitement TypeException1 > }
  catch ( TypeException2 E ) { <Traitement TypeException2 > }
  .....
  catch ( TypeExceptionk E ) { <Traitement TypeExceptionk > }
```

Où TypeException1, TypeException2, ... , TypeExceptionk sont des classes d'exceptions obligatoirement toutes **distinctes.** Seule une seule clause **catch** (TypeException E) {...}est exécutée (celle qui correspond au bon type de l'objet d'exception instancié).

Interception d'une ArrayStoreException :

```
class Action2 {
    public void meth(){
        // une exception est levée ...
    }
}
```

ArrayStoreException

```
class UseAction2{

    public static void main(String[] Args) {
        Action1 Obj = new Action1();
        System.out.println("Début du programme.");
        try{
            Obj.meth();
        }
        catch(ArithmeticException E){
            System.out.println("Interception ArithmeticException");
        }
        catch(ArrayStoreException E){
            System.out.println("Interception ArrayStoreException");
        }
        catch(ClassCastException E){
            System.out.println("Interception ClassCastException");
        }
        System.out.println("Fin du programme.");
    }
}
```

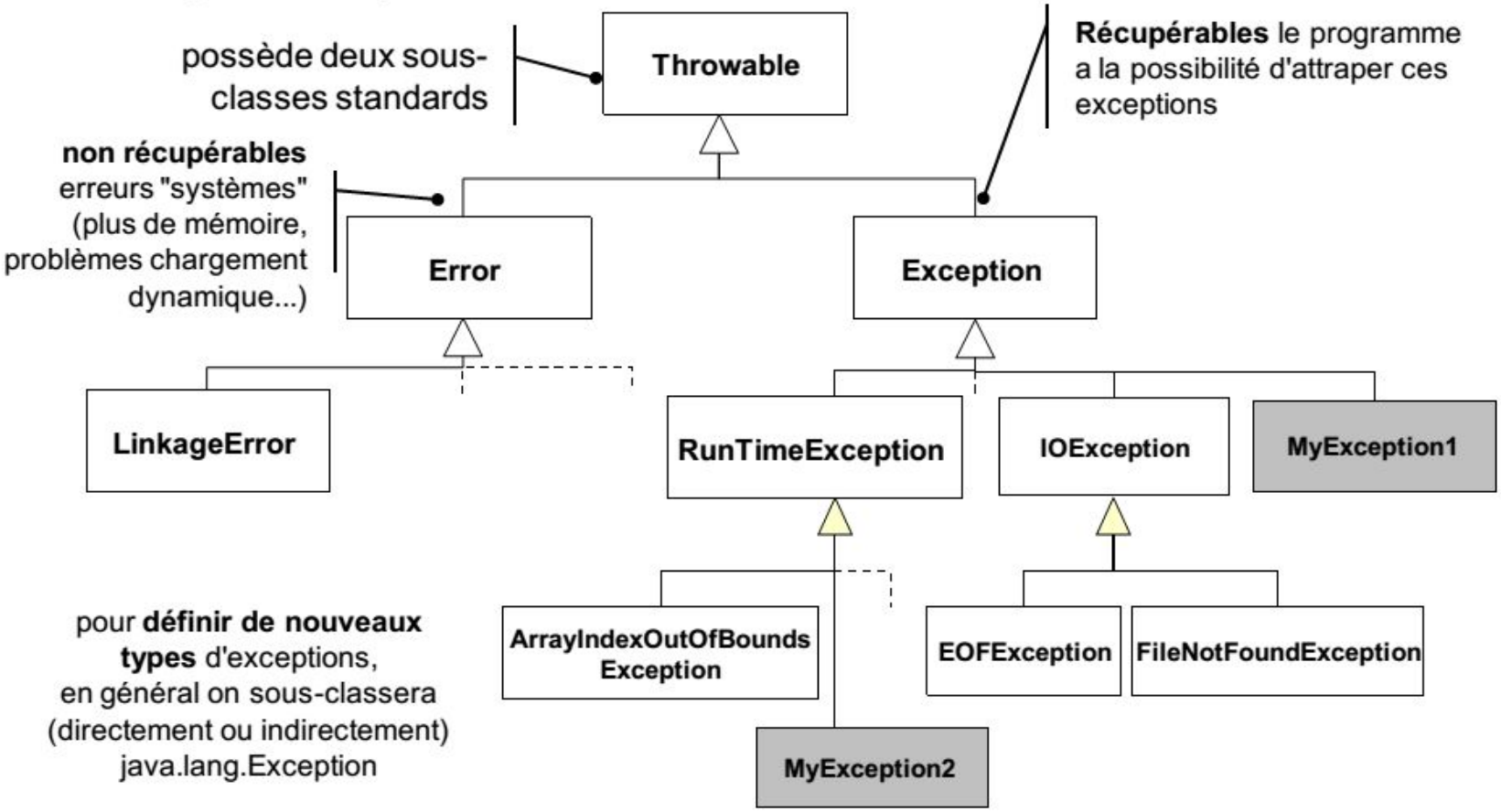
ArrayStoreException

poursuite du programme

EXCEPTIONS

OBJETS EXCEPTIONS

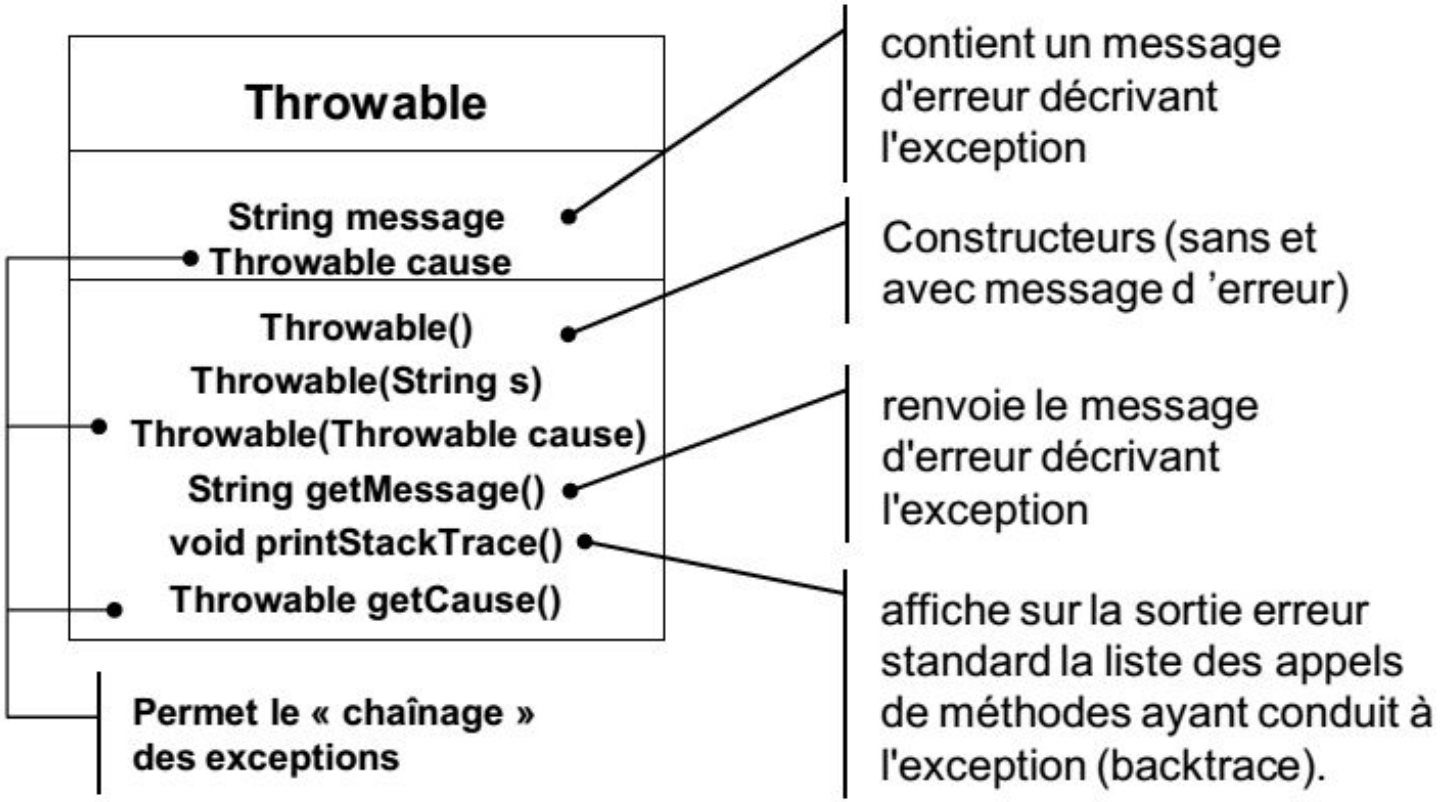
- en JAVA les **exceptions sont des objets**
  - *toute exception doit être une instance d'une sous-classe de la classe* ***`java.lang.Throwable`***



EXCEPTIONS

OBJETS EXCEPTIONS

- Puisqu'elles **sont des objets** les exceptions peuvent contenir :
  - *des attributs particuliers,*
  - *des méthodes.*
- Attributs et méthodes standards (définis dans **`java.lang.Throwable`**)



LES EXCEPTIONS PERSONNALISÉES

Ayant une classe Personne (décrite par un nom, prénom et âge) nous allons perfectionner un peu la gestion de nos objets Personne. Pour ce faire nous allons mettre en œuvre une exception afin d'interdire l'instanciation d'un objet personne avec un âge négatif.
Nous devons:
1. Créer une classe héritant de la classe Exception : AgeException (par convention, les exceptions ont un nom se terminant par « Exception ») ;
2. renvoyer l'exception levée à notre classe AgeException ;
3. ensuite, gérer celle-ci dans notre classe AgeException.

Pour faire tout cela, nous allons apprendre deux mots clés :
throws : ce mot clé permet de signaler à la JVM qu'un morceau de code, une méthode, une classe… est potentiellement dangereux et qu'il faut utiliser un bloc try{…}catch{…}. Il est suivi du nom de la classe qui va gérer l'exception.
throw : celui-ci permet tout simplement de lever une exception manuellement en instanciant un objet de type Exception (ou un objet hérité).

```
class AgeException extends Exception{
        public AgeException()
        {       System.out.println("Vous essayer d'instancier une personne avec un âge négatif!!");
        }
}
```

```
class Personne{
String nom;
String prenom;
int age;
public Personne(){nom=""; prenom=""; age=0;}
public Personne(String nom, String prenom, int age)
  {   this.nom = nom;
      this.prenom = prenom;
      this.age = age;
  }


public String toString(){
return ("la personne s'appelle: "+ nom+" "+prenom+" elle est agée de: "+ age+ " ans.");
}
}
```

C'est dans le constructeur de nos objets que nous allons ajouter une condition qui, si elle est remplie, lèvera une exception de type AgeException. En gros, nous devons dire à notre constructeur de Personne : « si l'utilisateur crée une instance de Personne avec un âge négatif, créer un objet de type AgeException ».

```
public Personne(String nom, String prenom, int age) throws  AgeException
 {
   if(age < 0)
     throw new AgeException();
   else
   { this.nom = nom;
     this.prenom = prenom;
     this.age = age;      }

 }
```

```
        static public  void main(String []args){
            Personne p= new Personne("Ben salah", "Ali", 12);
        }
    }
```

throws AgeException nous indique que si une erreur est capturée, celle-ci sera traitée en tant qu'objet de la classe AgeException, ce qui nous renseigne sur le type de l'erreur en question. Elle indique aussi à la JVM que le constructeur de notre objet Personne est potentiellement dangereux et qu'il faudra gérer les exceptions possibles.

Si la condition if (age < 0) est remplie, throw new AgeException(); instancie la classe AgeException. Par conséquent, si un age est négatif, l'exception est levée.

Cela signifie qu'à partir de maintenant, vu les changements dans le constructeur, il vous faudra gérer les exceptions qui pourraient survenir dans cette instruction avec un bloc try{…}catch{}.
Ainsi, pour que l'erreur disparaisse, il nous faut entourer notre instanciation avec un bloc try{…}catch{…}

```
static public  void main(String []args){
    try {
         Personne p= new Personne("Ben salah", "Ali", 12);
         System.out.println(p.toString());
    } catch (AgeException e) {  }

 }
```

Attention, l'instance de l' objet Personne a été déclarée dans le bloc try{…}catch{…} et cela peut causer beaucoup de problèmes. Par exp:

```
static public  void main(String []args){
    try {
         Personne p= new Personne("Ben salah", "Ali", 12);
         } catch (AgeException e) { }
    System.out.println(p.toString());

 }
```

ne fonctionnera pas, tout simplement parce que la déclaration de l'objet Personne est faite dans un sous-bloc d'instructions, celui du bloc try{…}. Une variable déclarée dans un bloc d'instructions n'existe que dans celui-ci. Ici, la variable p n'existe pas en dehors de l'instruction try{…}. Pour pallier ce problème, il nous suffit de déclarer notre objet en dehors du bloc try{…} et de l'instancier à l'intérieur :

```
static public  void main(String []args){
    Personne p=null;
    try {
          p= new Personne("Ben salah", "Ali", 12);
         } catch (AgeException e) { }
    System.out.println(p.toString());

 }
```

Mais que se passera-t-il si nous déclarons une Personne avec un âge négatif pour tester notre exception ? En remplaçant « 12» par « -12 » dans l'instanciation de notre objet ? C'est simple : en plus d'une exception levée pour l'âge négatif, vous obtiendrez aussi une NullPointerException.

Voyons ce qui s'est passé :

Nous avons bien déclaré notre objet en dehors du bloc d'instructions.

Au moment d'instancier celui-ci, une exception est levée et l'instanciation échoue !

La clause catch{} est exécutée : un objet AgeException est instancié.

Lorsque nous arrivons sur l'instruction « System.out.println(p.toString()); », notre objet est null !

Une NullPointerException est donc levée !

Ce qui signifie que si l'instanciation échoue dans notre bloc try{}, le programme plante ! Pour résoudre ce problème, on peut utiliser une simple clause finally avec, à l'intérieur, l'instanciation d'un objet Personne par défaut si celui-ci est null.

```
finally{
        if(p == null)
         p = new Personne();}
```

LA GESTION DE PLUSIEURS EXCEPTIONS

Supposons que nous voulons lever une exception si le nom fait moins de 3 caractères. Nous allons répéter les premières étapes vues précédemment, c'est-à-dire créer une classe NomException:

```
class NomException extends Exception {

   public NomException(String message){

  super(message);

 }

}
```

```
public Personne(String nom, String prenom, int age) throws  AgeException, NomException
  {
    if(age < 0)
      throw new AgeException();
    if(nom.length() < 3)
            throw new NomException("le nom est inférieur à 3 caractères ! nom = " + nom);
    else
    {

      this.nom = nom;
      this.prenom = prenom;
      this.age = age;

    }
  }
```

les différentes erreurs dans l'instruction throws sont séparées par une virgule. Nous sommes maintenant parés pour la capture de deux exceptions personnalisées. Regardez comment on gère deux exceptions sur une instruction :

```
static public  void main(String []args){
    Personne p=null;
    try {
          p= new Personne("Be", "Ali", 12);
         }
    //Gestion de l'exception sur l'âge
    catch (AgeException e) {e.printStackTrace();}
    //Gestion de l'exception sur le nom
    catch (NomException e) {System.out.println(e.getMessage());}
    finally{
            if(p == null)
             p = new Personne();}
    System.out.println(p.toString());

 }
```

Constatez qu'un deuxième bloc catch{} s'est glissé, c'est comme cela que nous gérerons plusieurs exceptions !
Si vous mettez un nom de moins de 3 caractères et un âge négatif, c'est l'exception de l'âge qui sera levée en premier, et pour cause : il s'agit de la première condition dans notre constructeur. Lorsque plusieurs exceptions sont gérées par une portion de code, pensez bien à mettre les blocs catch dans un ordre pertinent.

DEPUIS JAVA 7 : LE MULTI-CATCH…

il est possible de catcher plusieurs exceptions dans l'instruction catch. Ceci se fait grâce à l'opérateur « | » qui permet d'informer la JVM que le bloc de code est susceptible d'engendrer plusieurs types d'exception.

```
catch (AgeException | NomException e) {System.out.println(e.getMessage());}
```

Lorsqu'un événement que la JVM ne sait pas gérer apparaît, une exception est levée (exemple : division par zéro). Une exception correspond donc à une erreur.

La superclasse qui gère les exceptions s'appelle Exception.

Vous pouvez créer une classe d'exception personnalisée : faites-lui hériter de la classe Exception.

L'instruction qui permet de capturer des exceptions est le bloc try{…}catch{}.

Si une exception est levée dans le bloc try, les instructions figurant dans le bloc catch seront exécutées pour autant que celui-ci capture la bonne exception levée.

Vous pouvez ajouter autant de blocs catch que vous le voulez à la suite d'un bloc try, mais respectez l'ordre : du plus pertinent au moins pertinent.

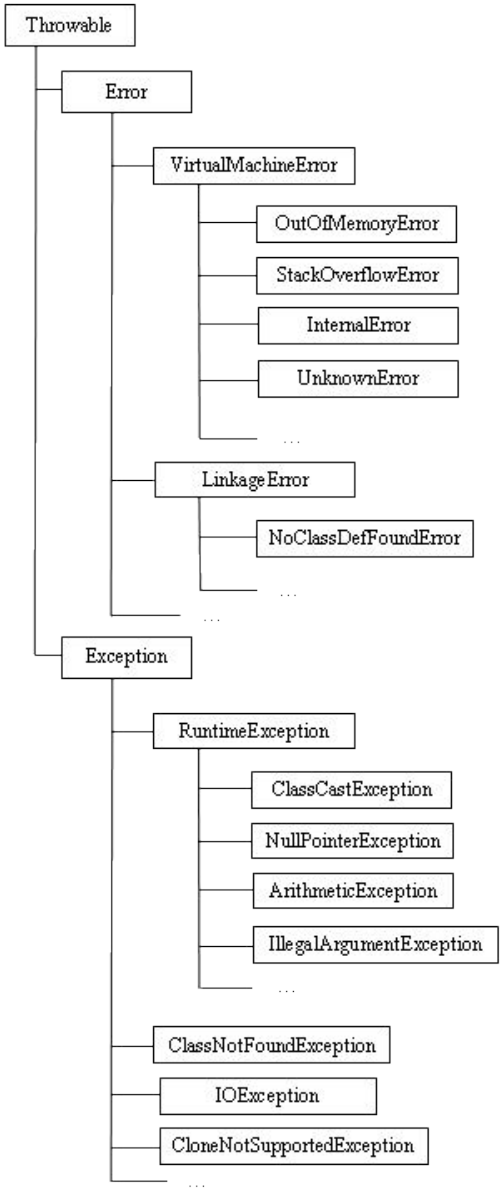
Dans une classe objet, vous pouvez prévenir la JVM qu'une méthode est dite « à risque » grâce au mot clé throws.

Vous pouvez définir plusieurs risques d'exceptions sur une même méthode. Il suffit de séparer les déclarations par une virgule.

Dans cette méthode, vous pouvez définir les conditions d'instanciation d'une exception et lancer cette dernière grâce au mot clé throw suivi de l'instanciation.

Une instanciation lancée par le biais de l'instruction throw doit être déclarée avec throws au préalable !

TYPES



RuntimeException :traitent des problèmes comme ceux-ci :

- un mauvais transtypage ;
- un accès à un tableau en dehors des limites ;
- l'emploi d'un pointeur null

pas obligé de les traiter. Erreur de prog.

Error : décrit les erreurs internes ou le manque de ressources dans le système d'exécution de Java. En principe, vous ne devez pas faire un throw d'un objet de ce type. On ne peut pas grand-chose dans le cas de ce type d'erreur, au demeurant assez rare, sauf à en informer l'utilisateur et tenter de mettre fin au programme de façon adéquate. Pas obligé de les traiter (rare de le faire). Erreur fatale.

Exception : Les exceptions n'héritant pas de RuntimeException comprennent :

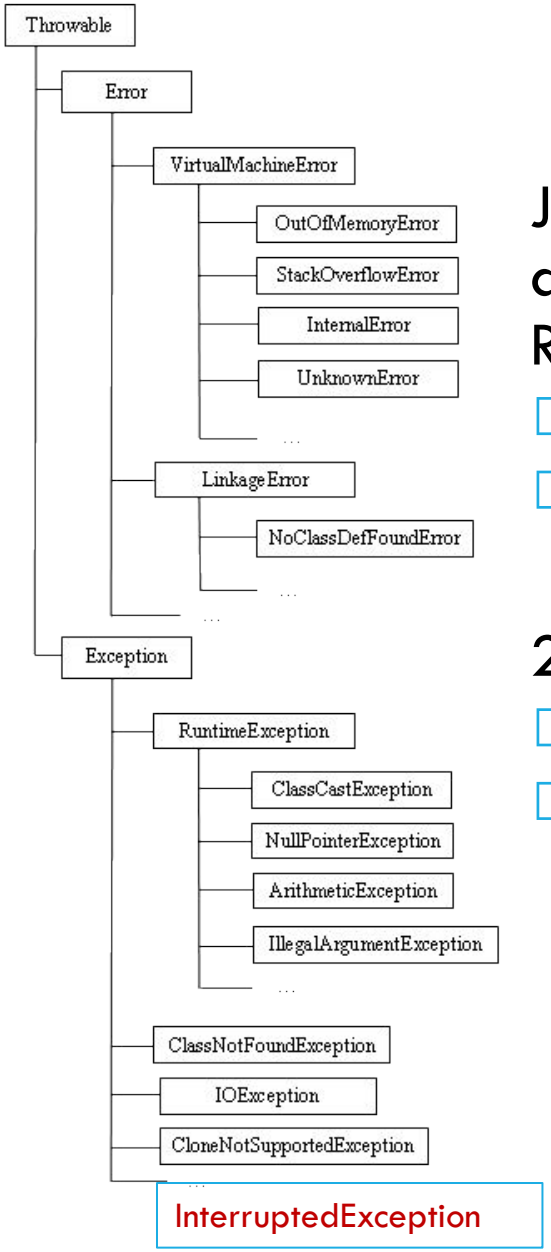
- lire au-delà de la fin d'un fichier ;
- accéder à une URL incorrecte ;
- rechercher un objet Class en donnant un nom ne correspondant à aucune classe existante.

obligé de les traiter. Erreur externe.

Problèmes :

- Exception ne représente pas toutes les ... exceptions !
- RuntimeException hérite d'Exception (donc catch(Exception) possible)

2 TYPES



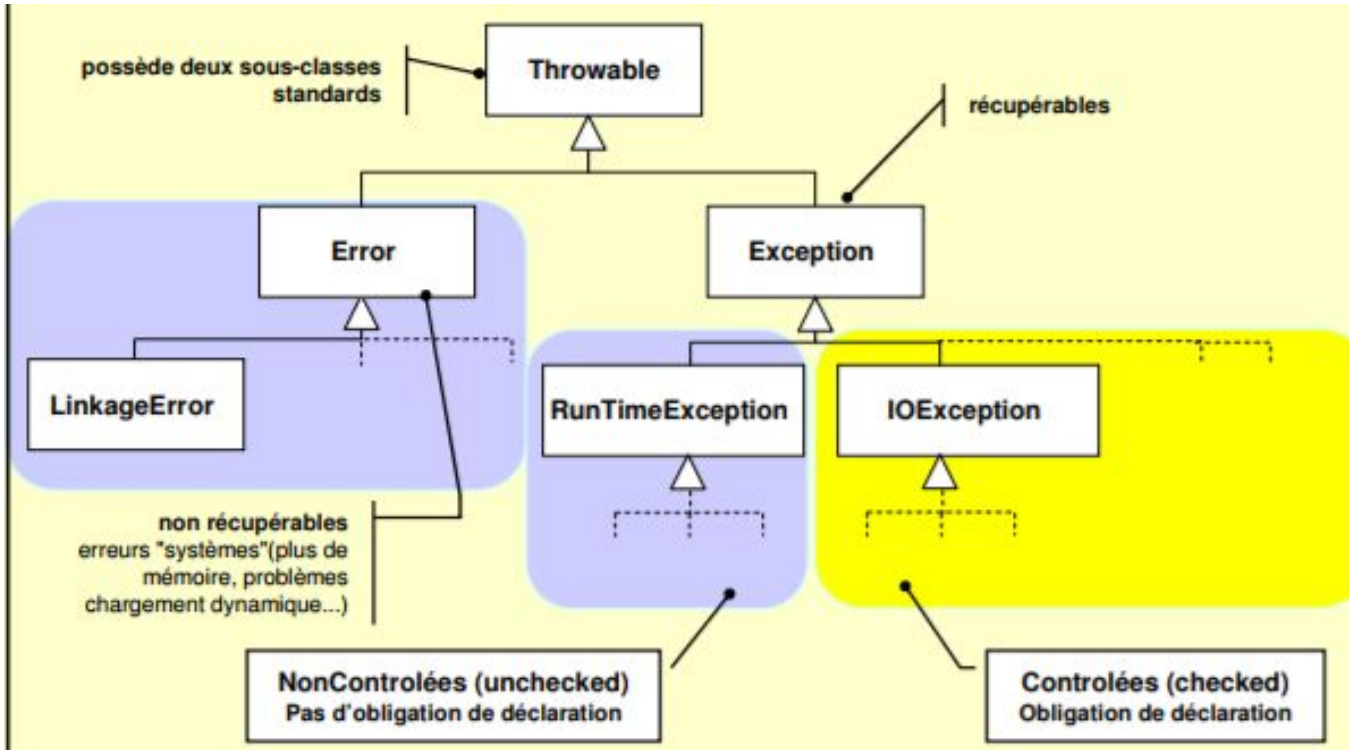
Java oblige à attraper les sous-types de Throwable qui ne sont pas des sous-types de Error ou RuntimeException (e.g. InterruptedException)

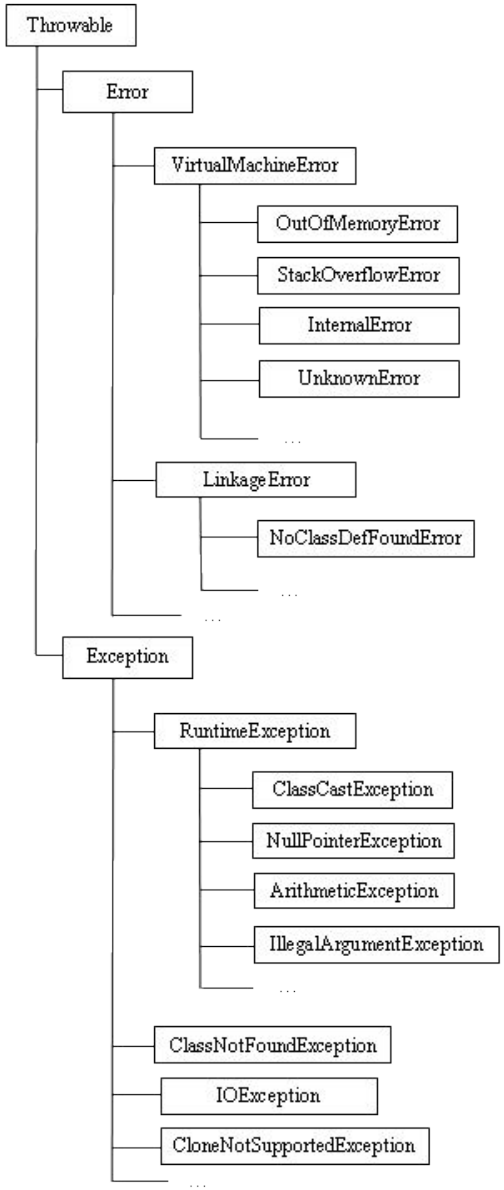
- Soit l'attraper avec try/catch.
- Soit dire à la méthode appelante de s'en occuper (throws) (penser à remplir le champ @throws de la javadoc !).

2 types :

- A attraper obligatoirement (checked).
- Les autres (non-checked).

CHECKED/UNCHEKED





Runtime exceptions

The java.lang package defines the following standard runtime exception classes:

ArithmeticException
This exception is thrown to indicate an exceptional arithmetic condition, such as integer division by zero.

ArrayIndexOutOfBoundsException
This exception is thrown when an out-of-range index is detected by an array object. An out-of-range index occurs when the index is less than zero or greater than or equal to the size of the array.

ArrayStoreException
This exception is thrown when there is an attempt to store a value in an array element that is incompatible with the type of the array.

ClassCastException
This exception is thrown when there is an attempt to cast a reference to an object to an inappropriate type.

IllegalArgumentException
This exception is thrown to indicate that an illegal argument has been passed to a method.

NegativeArraySizeException
This exception is thrown in response to an attempt to create an array with a negative size.

NullPointerException
This exception is thrown when there is an attempt to access an object through a null object reference. This can occur when there is an attempt to access an instance variable or call a method through a null object or when there is an attempt to subscript an array with a null object.

NumberFormatException
This exception is thrown to indicate that an attempt to parse numeric information in a string has failed.

RuntimeException
The appropriate subclass of this exception is thrown in response to a runtime error detected at the virtual machine level. Because these exceptions are so common, methods that can throw objects that are instances ofRuntimeException or one of its subclasses are not required to declare that fact in their throws clauses.