# EXCEPTIONS

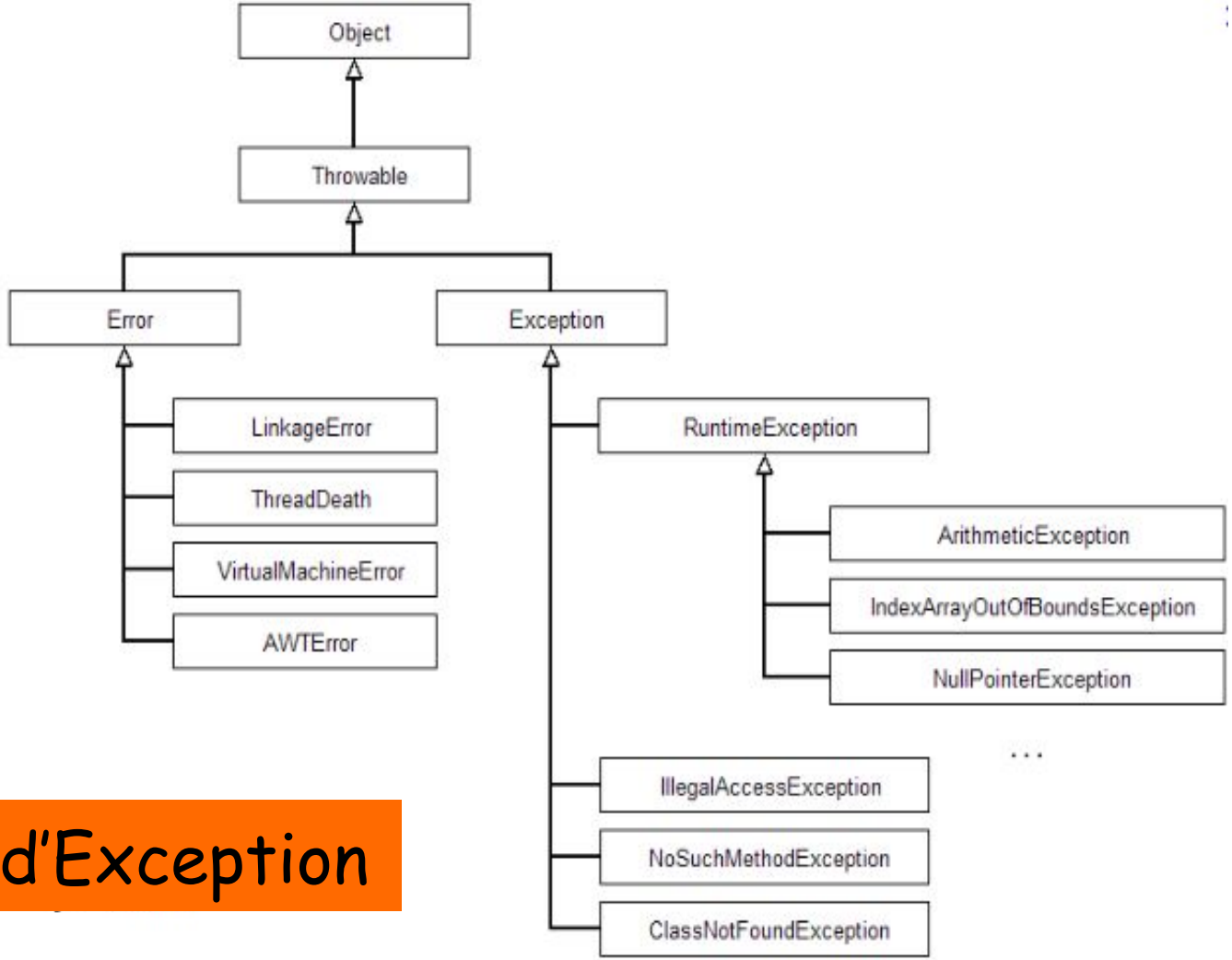
Martine GAUTIER - Université de Lorraine
martine.gautier@univ-lorraine.fr

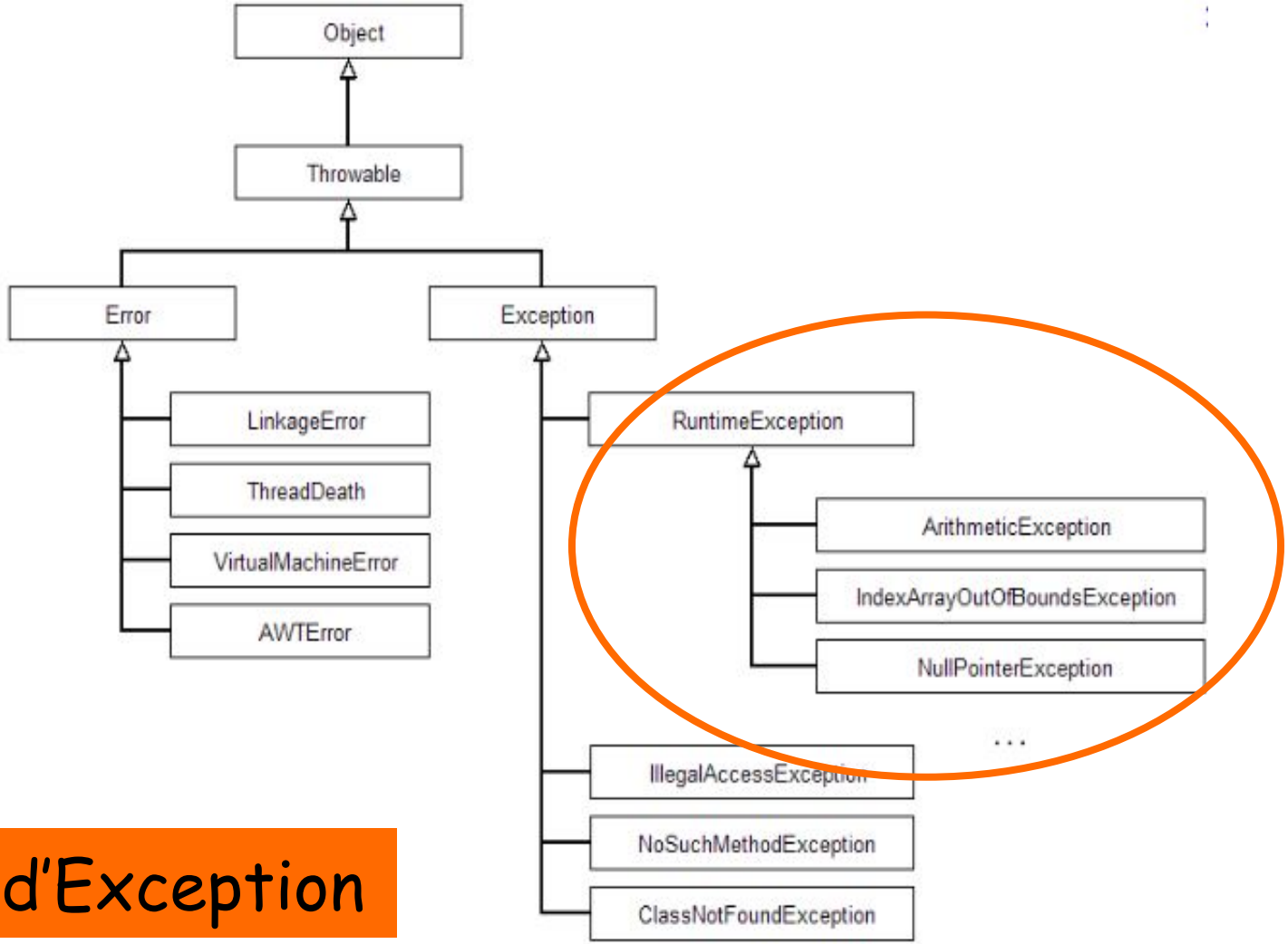
# Erreur à l'exécution

- Toute anomalie lors de l'exécution provoque le déclenchement d'une exception.
- Une exception est un objet, instance d'une sous-classe de `Exception`

  → `IndexOutOfBoundsException`, `NullPointerException`, *etc.*

# Hiérarchie d'Exception






# Hiérarchie d'Exception

# Traitement d'une RunTimeException

- Arrêt immédiat de l'exécution de la fonction en cours
- Retour à la fonction appelante, *etc.* jusqu'à la fonction `main`
- Transmission de l'exception de fonction en fonction.
- Affichage sur la sortie standard de la trace des différents appels

→ le programme *se plante* ….

# Subir le déclenchement d'une exception

Le programmeur débutant est très familier des exceptions …

```
class Test {
   private Point p ;
   public void test(Point pt) {
         pt.deplacer(1., 2.) ;
   }
…
   test(p) ;
}
```

provoque l'exception `NullPointerException`, si la variable `p` n'est pas affectée avant l'appel à la fonction `test`

# Programmer le déclenchement d'une exception

Instruction `assert` (avec option -ea) dans les tests

```
class TestOctet {
   public void testMasquer {
         Octet o, m ;
         o = new Octet(126);
         m = new Octet(1);
         o.masquer(m);
         assert o.toString().equals("0");
         assert m.toString().equals("1");
...
   }
...
}
```

# Programmer le déclenchement d'une exception

Instruction throw

```
class Pingouin {
   private int posx, posy ;
   public Pingouin(int px, int py) {
     if (px < 0 || py < 0)
        throw new IllegalArgumentException(“Paramètre incorrect“) ;
     posx = px ;
     posy = py ;
   }
...
}
```

# Programmer le déclenchement d'une exception

Instruction `assert` (avec option -ea) pour tester la validité des paramètres

```
class Pingouin {
   private int posx, posy ;
   /** @param px, py position du pingouin dans l'océan
    *  @exception AssertionError si l'un des paramètres est négatif
   */
   public Pingouin(int px, int py) {
     assert (px >= 0 && py >= 0):"Paramètre incorrect" ;
     posx = px ;
     posy = py ;
   }
...
}
```

# Traitement alternatif d'une RunTimeException

- L'arrêt brutal du programme n'est pas toujours la bonne réponse à l'anomalie.

  → l'erreur vient d'une donnée mal saisie, d'un fichier inexistant, *etc.*

  → effectuer une nouvelle saisie, ouvrir un autre fichier, *etc.*

- Mécanisme permettant d'interrompre la transmission de l'exception, et ainsi l'empêcher d'atteindre la fonction `main`

# Exemple avec NumberFormatException

Fonction Integer.parseInt(String s) pour convertir une chaîne en entier

```
class GD {
   protected int valeur ;
   public void setVal(String v) {
     this.valeur = Integer.parseInt(v) ;
   }
...
}
```

```
class UGD {
  protected GD gd ;

  public void setValGD(String v) {
     gd.setVal(v) ;
   }
...
}
```

```
UGD u ;

u.setValGD(“azerty”) ;
```

Exception déclenchée
NumberFormatException

# Exemple avec NumberFormatException

Fonction Integer.parseInt(String s) pour convertir une chaîne en entier

```
UGD u ;
String x = scanner.nextLine() ;
try { u.setValGD(x) ;
    …
    }
catch (NumberFormatException nfe) {
    System.out.println(“Donnée incorrecte”) ;
}
```

Capture de l’exception NumberFormatException

# Capturer deux exceptions ?

```
UGD u ;
String x = scanner.nextLine() ;
try { u.setValGD(x) ;
    …
    }
catch (NumberFormatException nfe) {
    System.out.println(nfe.getMessage()) ;
    Throw new IllegalArgument(“cfghjk”);
}catch (IllegalArgumentException iae) {
    System.out.println(“Argument incorrect”) ;
}catch (Exception e) {
    System.out.println(“Argument incorrect”) ;}
```

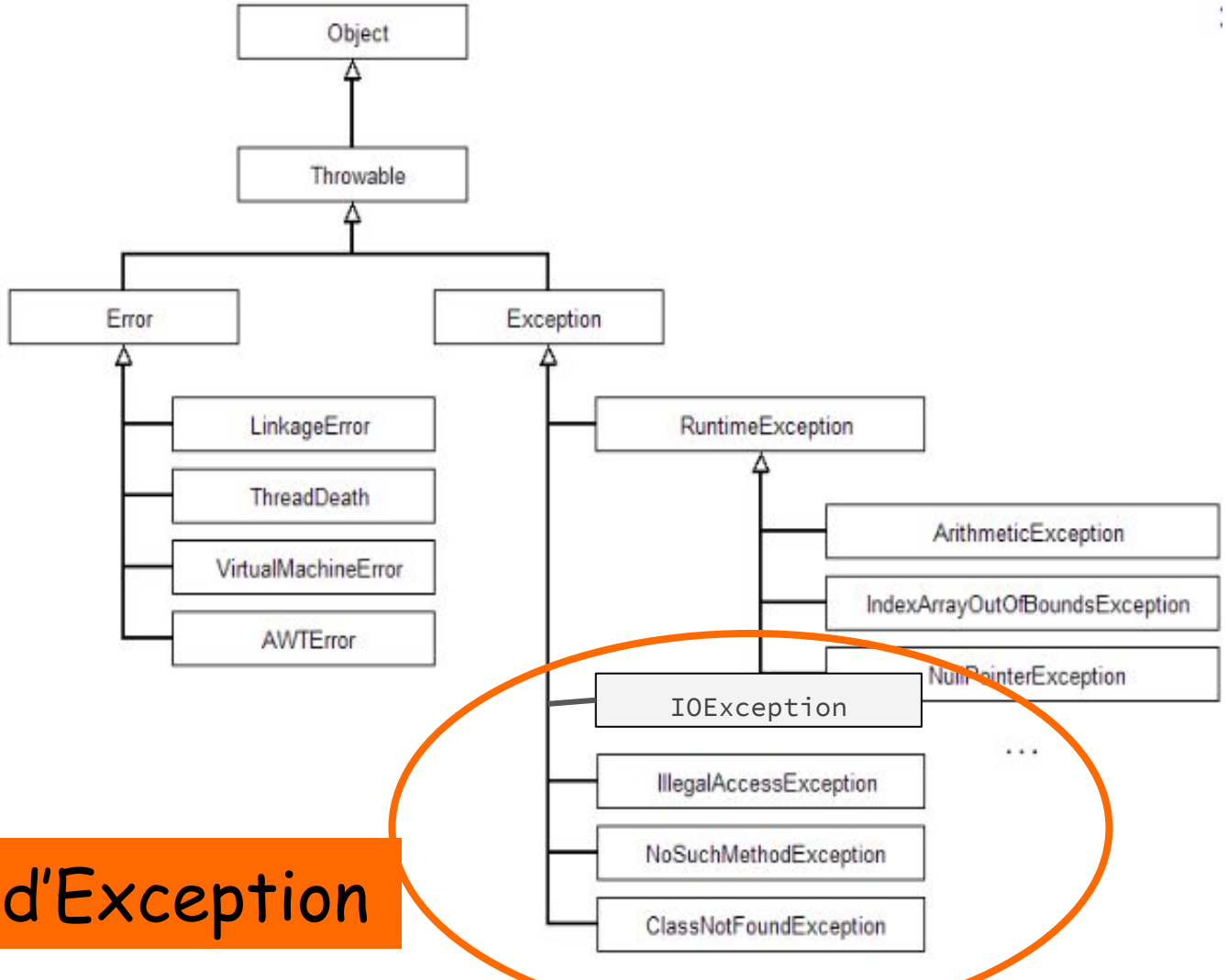
L'ordre des clauses catch a de l'importance

# Capturer toutes les exceptions ?

```
UGD u ;
String x = scanner.nextLine() ;
try { u.setValGD(x) ;
    …
    }
catch (NumberFormatException nfe) {
    System.out.println(“Donnée incorrecte”) ;
catch (NullPointerException npe) {
    System.out.println(“Bug”) ;
catch (ArithmeticException ae) {
    System.out.println(“Bug”) ;
```

# Traitement d'une autre exception

- Pas de traitement par défaut pour les autres exceptions
- Préciser, pour chaque fonction, si l'exception est transmise à la fonction appelante ou bien capturée




# Hiérarchie d'Exception

## createTempFile

```
public static File createTempFile(String prefix,
                                  String suffix)
                           throws IOException
```

Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name. Invoking this method is equivalent to invoking `createTempFile(prefix, suffix, null)`.

The `Files.createTempFile` method provides an alternative method to create an empty file in the temporary-file directory. Files created by that method may have more restrictive access permissions to files created by this method and so may be more suited to security-sensitive applications.

**Parameters:**

`prefix` - The prefix string to be used in generating the file's name; must be at least three characters long

`suffix` - The suffix string to be used in generating the file's name; may be `null`, in which case the suffix `".tmp"` will be used

**Returns:**

An abstract pathname denoting a newly-created empty file

**Throws:**

`IllegalArgumentException` - If the `prefix` argument contains fewer than three characters

`IOException` - If a file could not be created

`SecurityException` - If a security manager exists and its `SecurityManager.checkWrite(java.lang.String)` method does not allow a file to be created

RunTimeException
& Exception

# Exemple avec IOException

Fonction `File.createTemp(String s1, String s2)`

```
File f = File.createTemp(nom, suffixe)  ;
```

Le compilateur indique que l'exception doit être capturée ou transmise.

# Exemple avec IOException (capturée ou transmise)

```
void uneFonction(String nom, String suffixe)
  try {
    File f = File.createTemp(nom, suffixe)  ;
    …
    }
  catch (IOException nfe) {
    System.out.println(“Impossible de créer le fichier”) ;
}
```

capturée

```
void uneFonction(String nom, String suffixe)
                                        throws IOException {
    File f = File.createTemp(nom, suffixe)  ;
    …
}
```

transmise à la fonction appelante, qui devra aussi choisir

# De nouvelles exceptions

Possible de se définir sa propre hiérarchie d'héritage sous `Exception`

```
public class MonExceptionPerso extends Exception {
  public MonExceptionPerso(String message) {
     super(message) ;
  }
}
```

Juste un constructeur à écrire

```
if (x>=100) throw new MonExceptionPerso("Valeur trop grande") ;
```

Déclenchée comme une autre

```
try { … }
catch (MonExceptionPerso mep) {
   System.out.println(mep.getMessage());
}
```

Capturée comme une autre

On hérite de la fonction `getMessage`

# De nouvelles exceptions, en cas d'héritage

La clause `throws` n'est pas héritée ; il faut la réécrire.

```
public class UneClasse   {
  public abstract void uneFonction() throws MonExceptionPerso ;
  public void uneAutreFonction() throws MonExceptionPerso {
  ....
  }
}
```

```
public class UneSousClasse extends UneClasse  {
  public void uneFonction() throws MonExceptionPerso {
     ... ;
  }
  public void uneAutreFonction() throws MonExceptionPerso {
  ....
  }
}
```

# Pour conclure

- Le mécanisme d'exception permet de gérer facilement des situations anormales.
- Construire une hiérarchie d'exceptions permet de distinguer ces situations et de leur associer un traitement spécifique

- Ne pas remplacer toutes les conditionnelles par des exceptions !