# 0-M02 – Introduction à la programmation
# Introduction à l'algorithmique

## Cours n°5
## Gestion des exceptions

Camille Coti
camille.coti@iutv.univ-paris13.fr

IUT de Villetaneuse, département R&T

2011 – 2012

# Principe des exceptions

Remonter les erreurs rencontrées par une fonction à l'appelant

- Simplification du code
- Gestion des erreurs dans une classe séparée

Pour cela une méthode qui rencontre une erreur **lève une exception** qui est traitée par l'appelant.

Types d'exceptions :

- Erreurs graves : provoquent l'arrêt du programme
  - Classe `java.lang.Error`
- Erreurs devant être traitées
  - Classe `java.lang.Exception`
- Erreurs ne devant pas nécessairement être traitées
  - Classe `java.lang.RuntimeException` (hérite de `java.lang.Exception`)

# Mécanisme

On définit une méthode qui effectue une opération "à risque" (débordement de tampon, valeurs indéfinies, dépassement de limites d'un tableau...)

- Si quelque chose se passe mal : la méthode **lève une exception**

L'appelant **attrappe cette exception**

- Possibilité de traiter l'erreur, de la faire remonter, de faire autre chose...

Plusieurs types d'exceptions possibles pour une seule méthode

- Trairement selon l'erreur rencontrée
- On récupère des informations sur l'exception levée

# Méthode qui lève une exception

On déclare qu'une méthode va **lever** une exception

- Mot-clé **throws** + type d'exception dans la déclaration de la méthode

Exemple :

```
1 public int factorielle( int n ) throws PasDefini {
2   /* ... */
3 }
```

La méthode factorielle peut lever l'exception `PasDefini` en cas de problème ($n!$ non défini pour $n < 0$)

# Levée d'exception

Lorsque quelque chose ne se passe pas bien

- On **lève** l'exception
- Mot-clé **throw** + instanciation de l'exception

```
1  public int factorielle( int n ) throws PasDefini {
2    if( n < 0 ) {
3      throw new PasDefini();
4    } else {
5      int i, res;
6      res = 1;
7      for( i = 1 ; i <= n ; i++ ) {
8        res *= i;
9      }
10     return res;
11   }
12 }
```

# Attrappage d'exception

On utilise la syntaxe **try ... catch**

- Bloc **try** : exécution des instructions qui peuvent lever une exception
- Bloc(s) **catch** : comportement suivi si une exception est levée

```
1  int nb, res;
2  System.out.println( "Saisir un nombre" );
3  nb = Clavier.lireInt();
4  try {
5    res = factorielle( nb );
6  }
7  catch( PasDefini e ) {
8    System.out.println( "Valeur non definie" );
9  }
```

Si plusieurs exceptions possibles : plusieurs blocs `catch`

# Créer un nouveau type d'exception

Suivant le type d'erreur, l'exception hérite d'une classe mère

- Classe `java.lang.Error`, `java.lang.Exception` ou `java.lang.Exception`

Elles héritent toutes de l'interface `Throwable`

- Exemple de méthode disponible : `printStackTrace()`

Exemple :

```
1  class PasDefini extends Exception {
2    public int nb;
3    public PasDefini() {
4      System.out.println( "Nombre indefini" );
5    }
6    public PasDefini( int n ) {
7      nb = n;
8      System.out.println( n + "indefini" );
9    }
10 }
```

# Exemples d'exceptions courantes

## Héritent de java.lang.Error

- `TypeMismatchException`
- `IOError`

## Héritent de java.lang.Exception

- `IllegalAccessException`
- `ClassNotFoundException`
- `InterruptedException`

## Héritent de java.lang.RuntimeException

- `NullPointerException`
- `ArithmeticException`
- `ClassCastException`
- `IndexOutOfBoundsException`