Les exceptions

"Les généralités ne sont constituées que d'exceptions."
Solal Malibu

Exception _ définition

- Exception: situation anormale détectée
- On dit qu'une méthode ayant détecté une situation anormale déclenche (*throws*) une exception. Cette exception pourra être capturée (*catch*) par le code.
- Méthode déclenche exception -> JVM remonte l'invocation des méthodes jusqu'à trouver une méthode qui capture cette exception. Sinon exécution arrêtée.

Exceptions

- De nombreuses fonctions Java peuvent générer des exceptions
- Obligation à gérer ces exceptions
  ⇒ programmes plus résistants aux erreurs
- Principe

```
try{
      fonction pouvant générer une exception
}catch(Exception e){
     gestion de l'exception
}
finally{...}
instruction suivante
```


Exceptions _ mots clés

- Mot clé **try**
  - permet de spécifier une section de code sur laquelle une exception peut être levée
- Mot clé **catch**
  - sert à spécifier le code à exécuter pour une exception
- Mot clé **finally**
  - permet d'introduire un code de traitement qui sera de toute manière exécuté (après *try* ou *catch*)
  - facultatif

Exceptions _ mots clés (ex)

```
public static int test(int tab[], int indice){
    try{
        return (tab[indice]);
    }catch(Exception ex){
        return (0);
    }finally{
        System.out.println("bloc finally");
    }
}
```

```
public static void main(String[] args) {
    int t[]=new int[2];
    t[0]=1;t[1]=2;
    int res=test(t,1);
    System.out.println("res="+res);
    res=test(t,5);
    System.out.println("res="+res);
}
```

```
run:
bloc finally
res=2
ı thread "main"
bloc finally
rayIndexOutOfBoundsException
res=0
```


Exceptions _ classes

- *Exception* est une classe mère (dérivée de *Throwable*) de plusieurs autres classes
  - IOException
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - RuntimeException
  - …
- Il est possible d'effectuer différentes tâches en fonction du type d'exception

7

Exceptions _exemple

```
public static void main(String[] args) {
int t[]=new int[3];
int i,j,entier;
for(i=0;i<args.length;i++){
  try{
      entier=Integer.parseInt(args[i]);
      System.out.println(entier);
      for(j=0;j<entier;j++)
         t[j]=0;
  }catch(NumberFormatException e){
      System.out.println("Argument "+(i+1)+" non entier");
  }
  catch(ArrayIndexOutOfBoundsException ee){
      System.out.println("indice hors limites");
  }
} }
```

```
Arguments fournis: 2 texte 12 15.5

Exécution:
2
Argument 2 non entier
12
indice hors limites
Argument 4 non entier
```

8

Exceptions

- Méthodes utiles
  - .getMessage(): renvoie un message détaillant l'erreur qui s'est produite
  - .toString(): indique le type de l'exception ainsi que la valeur de la propriété *Message*
- Créer ses propres exceptions

```
public class MyException extends Exception{

    public MyException(String message){
          super(message)      }
    …
}
```

9

Lancer une exception

```
public static int test2(int tab[], int indice)  throws  Exception {
      if(indice>=0 && indice<tab.length)
          return (tab[indice]);
      else
          throw new Exception("erreur d'indice);
  }
```

```
public static int test(int tab[], int indice){
   try{
      return (tab[indice]);
   }catch(Exception ex){
      return (0);
   }
}
```

```
public static void main(String[] args) {
    int t[]=new int[2];
    t[0]=1;t[1]=2;
    int res;

    res=test(t,5);        // res=0

    try {
          res = test2(t, 5);
       } catch (Exception ex) {
          System.out.println(ex);
       }            //affiche: erreur d'indice
}
```