## Gestion des erreurs

Les **exceptions** permettent d'*anticiper les erreurs* qui pourront potentiellement se produire lors de l'utilisation d'une portion de code.

Exemple : on veut écrire une fonction qui calcule l'inverse d'un nombre réel quand c'est possible :

| **f** |
| --- |
| entrée : $x$<br>sortie : $1/x$ |
| ***Si*** $x = 0$<br>*erreur*<br>***Sinon***<br>*retourner 1/x* |

☞ Mais que faire **concrètement** en cas d'erreur ?

## Gestion des erreurs (2)

① retourner une valeur choisie à l'avance :

```
double f(double x) {
  if (x != 0.0) {
    return 1.0 / x;
  } else {
    return Double.MAX_VALUE;
  }
}
```

Mais cela

1. **n'indique pas** à l'utilisateur potentiel qu'il a fait une erreur
2. retourne de toutes façons un **résultat inexact** ...
3. suppose une **convention arbitraire** (la valeur à retourner en cas d'erreur)

## Gestion des erreurs (3)

② afficher un message d'erreur :

```
double f(double x) {
  if (x != 0.0) {
      return 1.0 / x;
  } else {
    System.out.println("Erreur dans f : division par 0");
    return ????;
  }
}
```

mais que retourner effectivement en cas d'erreur ?...
...on retombe en partie sur le cas précédent

De plus, cela est **très mauvais** car produit des *effets de bord* : affichage dans le terminal alors que ce n'est pas du tout dans le rôle de `f` !

## Gestion des erreurs (4)

③ retourner un code d'erreur :

```
boolean f(double x, Double resultat) {
  if (x != 0.0) {
      resultat = 1.0 / x;
      return true;
  } else {
      return false;
  }
}
```

Cette solution est déjà **meilleure** car elle laisse à la fonction qui appelle `f` le soin de décider quoi faire en cas d'erreur.
Cela présente néanmoins l'inconvénient d'être assez lourd à gérer pour finir :

- cas de l'appel d'appel d'appel.... ...d'appel de fonction,
- mais aussi écriture peu intuitive :
  `if (f(x,y))... // le résultat de la division est dans y`
  au lieu de
  `y=f(x);`

## Exceptions

Il existe une solution permettant de *généraliser* et d'*assouplir* cette dernière solution : déclencher une **exception**

☞ mécanisme permettant de *prévoir une erreur* à un endroit
et de **la gérer à un autre endroit**

## Exceptions (2)

Principe :

- lorsque qu'une erreur a été détectée à un endroit, on la signale en *« lançant » un objet* contenant toutes les informations que l'on souhaite donner sur l'erreur
  (« lancer » = créer un objet disponible pour le reste du programme)
- à l'endroit où l'on souhaite gérer l'erreur (au moins partiellement), on peut *« attraper » l'objet « lancé »* (« attraper » = utiliser)
- si un objet « lancé » n'est pas attrapé du tout, cela provoque l'arrêt du programme : *toute erreur non gérée provoque l'arrêt*.

Un tel mécanisme s'appelle « gestion des exceptions ».

## Syntaxe de la gestion des exceptions

On cherche à remplir 4 tâches élémentaires :

1. signaler une erreur
2. marquer les endroits réceptifs aux erreurs
3. leur associer (à chaque endroit réceptif) un moyen de gérer les erreurs qui se présentent
4. éventuellement, « faire le ménage » après un bloc réceptif aux erreurs

On a donc 4 mots du langage Java dédiés à la gestion des exceptions :

`throw` : indique l'erreur (i.e. « lance » l'exception)
`try` : indique un bloc réceptif aux erreurs
`catch` : gère les erreurs associées (i.e. les « attrape » pour les traiter)
`finally` : (optionel) indique ce qu'il faut faire après un bloc réceptif

## Syntaxe de la gestion des exceptions

Notez bien que :

- L'indication des erreurs (`throw`) et leur gestion (`try`/`catch`) sont le plus souvent *à des endroits bien séparés* dans le code
- Chaque bloc `try` possède son/ses `catch` associé(s)

## Pour résumer

Une exception est un moyen de signaler un événement nécéssitant une attention spéciale au sein d'un programme, comme:

- une **erreur grave**
- une **situation inhabituelle** devant être traitées de façon particulière

☞ But : **améliorer la robustesse des programmes** en :

- séparant le code de traitement des erreurs du code « effectif »
- fournissant le moyen de forcer une réponse à des erreurs particulières

## Exceptions : intérêt

Avantages de la gestion des exceptions par rapports aux codes d'erreurs retournés par des fonctions :

- écriture plus facile, plus intuitive et plus lisible
- propagation **automatique** de l'exception aux niveaux supérieurs d'appel (fonction appelant une fonction appelant ...)
  plus besoin de gérer obligatoirement l'erreur au niveau de la fonction appelante
- une erreur peut donc se produire à n'importe quel niveau d'appel, elle sera toujours reportée par le mécanisme de gestion des exceptions

**Note :** si une erreur peut être gérée localement, le faire et ne pas utiliser le mécanisme des exceptions.)