# Les exceptions

# Les exceptions

- Outil de traitement des erreurs
- Héritage de différentes erreurs. 2 types d'erreurs :
  - `Error` : Erreurs graves. Stop immédiatement l'exécution (pas de traitement)
  - `Exception` : Erreurs moins graves. A traiter pour le bon fonctionnement du programme.

# Levées d'exceptions

On dit qu'une exception est **levée**.

Si une exception existe déjà dans java, la levée est automatique (ex : division par 0)

Si c'est une exception personnalisée, il faut écrire une instruction de levée :

```
throw new MonException();
```

ou

```
throw new MonException("message");
```

# Captures d'exceptions

Quand une exception est levée, il faut la **capturer** pour la traiter. Pour ça on utilise le `try{...}catch{...}`

```
try{
    // suite d'instructions qui peuvent
    // faire lever une exception
}catch (ExceptionPrecise e){
    // Traitement de l'erreur précise
}catch (ExceptionGenerale e){
    // Traitement de l'erreur générale
}finally{
    // Code à executer même si on sort
    // par une erreur.
}
```

# Propagation

Dans certains cas, il ne faut pas capturer les erreurs là où elles apparaissent.

```
public void affiche(String chaine) throws ChaineVideExc{
    if(chaine == ""){
        throw new ChaineVideExc("message");
    }else{
        System.out.println(chaine);
    }
```

```
public void input(String chaine){
    try{
        affiche(chaine);
    }catch (ChaineVideExc e){
        System.out.println(e);
```

# Exception personnalisée

On peut créer des exceptions selon les besoins de nos programmes (ex : si on veut un nombre positif)

```
class MonException extends Exception{
    public MonException(){
        super();
    }

    public MonException(String message){
        super(message);
    }

    public String toString(){
        this.getMessage();
    }
}
```

# Gestion des fichiers

# Classes utiles

- `File` : Fichier abstrait
- `FileInputStream` : Flux d'importation (codes de caractères).
- `FileOutputStream` : Flux d'exportation (codes de caractères).
- `DataInputStream` : Flux d'importation de données typées.
- `DataOutputStream` : Flux d'exportation de données typées.

# Lecture de fichier

```
try{
   File file = new File("fichier.txt");
   FileInputStream stream = new FileInputStream(file);
   DataInputStream input = new DataInputStream(stream);
   while (true) {
      System.out.print(input.readUTF());
   }

} catch (EOFException e) {
   System.out.println("Fin du fichier");
} catch (FileNotFoundException e) {
   System.out.println(e);
} catch (IOException e) {
   System.out.println(e);
}finally{
   output.close();
}
```

# Ecriture d'un fichier

```
try{
   String content = "Contenu du fichier";
   File file = new File("fichier.txt");
   FileOutputStream stream = new FileOutputStream(file);
   DataOutputStream output = new DataOutputStream(stream);

   output.writeUTF(content);

} catch (FileNotFoundException e) {
   System.out.println(e);
} catch (IOException e) {
   System.out.println(e);
}finally{
   output.close();
}
```

# Pour approfondir

1. RTFM
2. Tester son code
3. Poser des questions