Les exceptions

Les exceptions : principe

La gestion des erreurs se fait souvent par le renvoi d'un code d'erreur :

```
int methode(){
        int ok ;
        ok = sousMethode() ;
        if( !ok )          {            ...
                return 0 ;
        } else {
                ...
        }
}
```

- des variables sont mobilisées pour les erreurs ;
- le retour est utilisé pour les erreurs ;
- le code d'erreur doit être propagé à travers la pile des appels des fonctions.

Exemple d'utilisation d'exception

La gestion des erreurs de manipulation de fichiers :

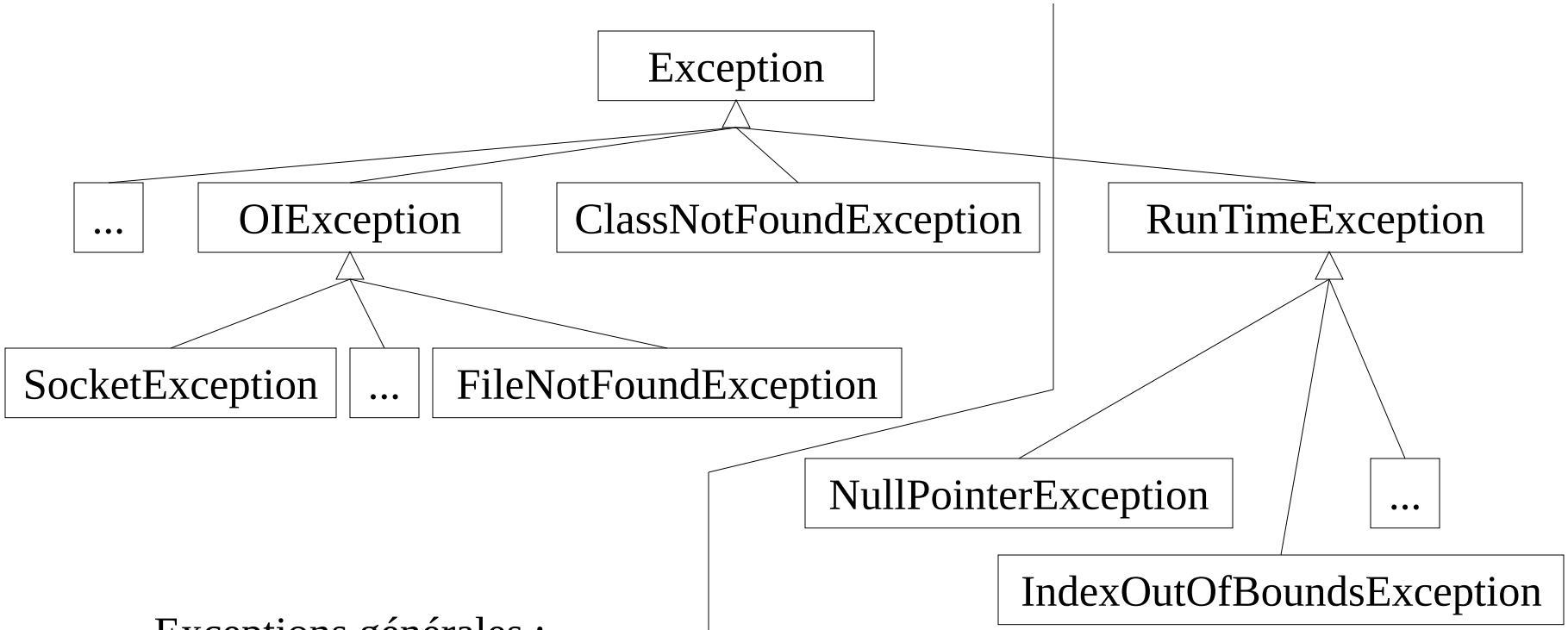
```
try{
        FileInputStream file = new FileInputStream ("essai.txt");
        ...
} catch(FileNotFoundException e){
            e.printStackTrace();
}
```

Les exceptions : règle

Si une méthode qui lance une exception est invoquée, il est possible de :

- capturer l'exception et la traiter (try … catch) ;
- capturer l'exception (try … catch) et lui associer une nouvelle exception (throws … throw) ;
- laisser passer l'exception (throws obligatoire).

Les exceptions pré-définies



Exceptions générales :

Le compilateur oblige à prendre en compte ces exceptions.

Exceptions à l'exécution :

Le compilateur n'oblige pas à prendre en compte ces exceptions.

Créer son exception

```
public class TestDException{

   void sousMethode() throws MonException{
        if( /*erreur*/ ) throw new MonException( "erreur" ) ;   // lance exception
        …
   }

   void methode() throws MonException{
        sousMethode() ;
   }

   public static void main( String[] argv ){
        TestDException test = new TestDException() ;
        try  {
             test.methode() ;       // essai
        }
        catch( MonException e ){            // erreur interceptée
             System.out.println( e.getMessage() ) ;
        }
   }
}
```

Créer un type d'exceptions :

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

- Il n'y a plus de variable pour gérer les erreurs ;
- le retour n'est plus utilisé pour les erreurs ;
- remontée automatique de l'erreur jusqu'à son traitement.

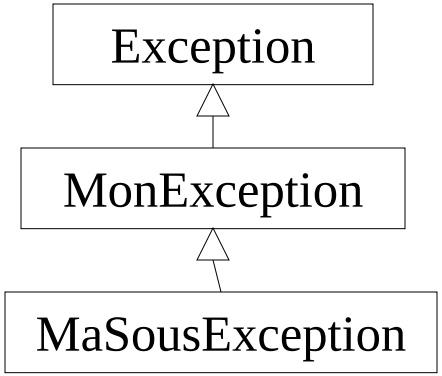
Créer son exception

```
public class TestDException{

        void sousMethode() throws MonException{
             if( /*erreur*/ ) throw new MonException( "erreur" ) ;   // lance exception
             …
        }

        void methode() throws MonException{
              SousFonction() ;
        }

        public static void main( String[] argv ) throws MonException{
             TestDException test = new TestDException() ;
              test.methode() ;
        }
}
```

- Même les erreurs ignorées provoquent la fin de l'exécution du programme.

Spécialiser les exceptions

- On peut spécialiser des exceptions afin d'ajouter des informations supplémentaires.



- S'il y a plusieurs catch, ce doit être du plus spécifique au plus général (sinon erreur).

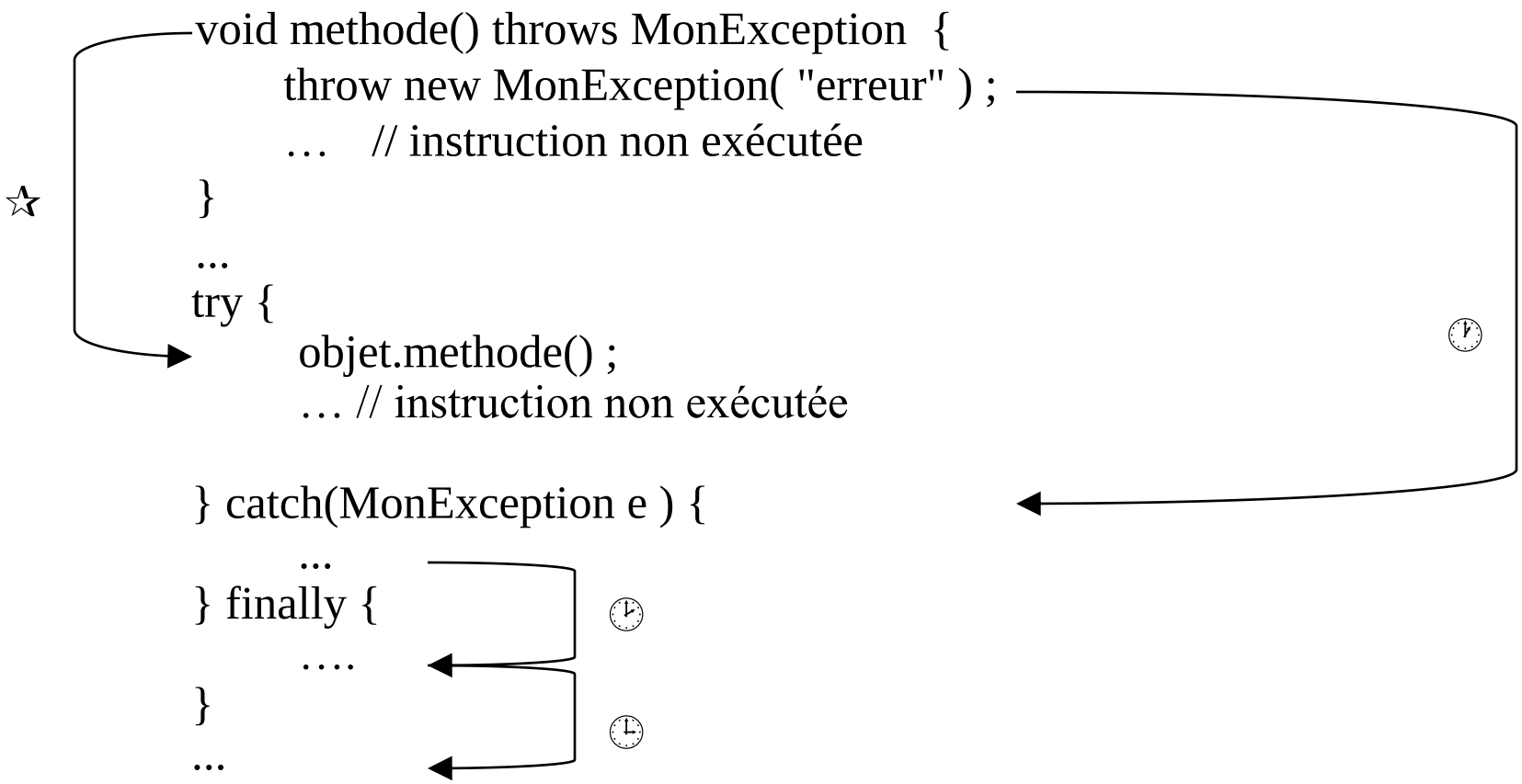
```
TestDException testException = new TestDException() ;
try{
      monException.methode() ;
}
catch( MaSousException e ) {        // le plus spécifique est avant
   // ...
}
catch( MonException e )    {        // le plus général
            // …
}
```

La clause finally

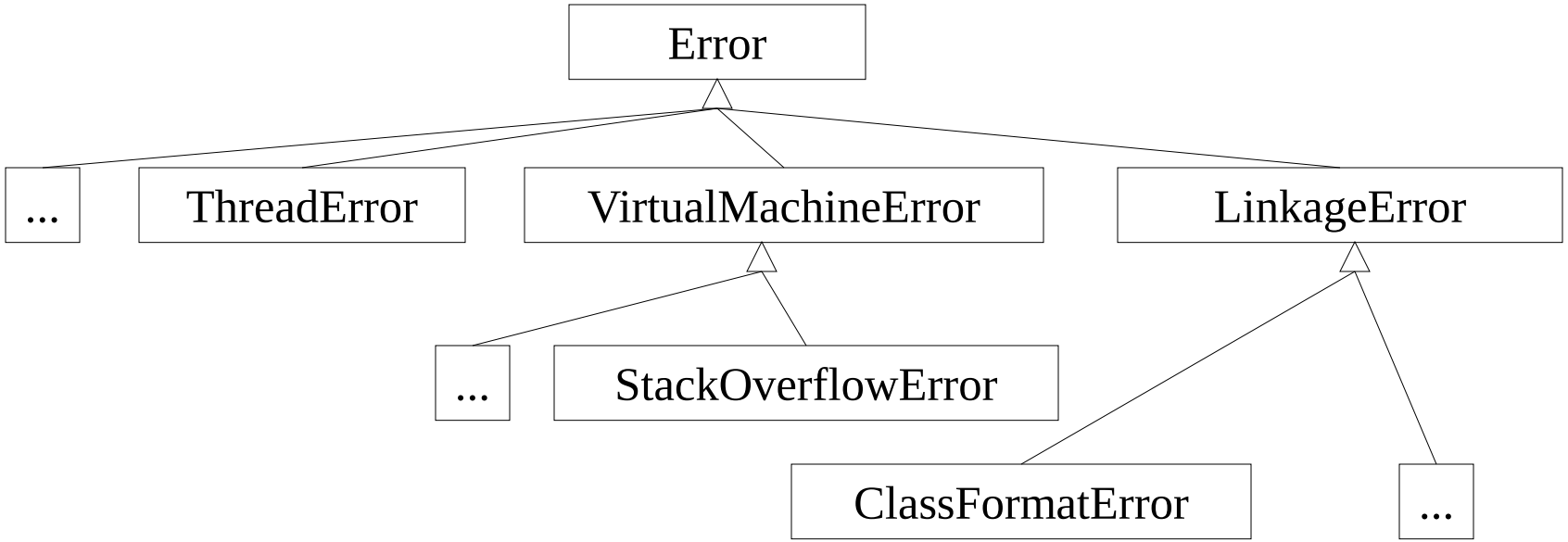
- La clause finally est toujours exécutée, qu'une exception ait été levée ou pas, ou qu'une instruction break ou return ait été rencontrée ;
- Utilisée pour garantir la libération de ressource qui ne sont pas des objets.

```
// recherche d'une ligne dans un fichier
public boolean chercher( String fichier, String ligne) throws IOException {
        BufferedReader in = null ;
        try {
                BufferedReader in = new BufferedReader( new FileReader( fichier ) );
                String line ;
                while( (line=in.readLine()) != null )    // une exception peut être levée
                        if( line.equals( ligne ) )              // une exception peut être levée
                                return true ;
                return false ;
        } finally {
                if( in != null )
                        in.close() ;     // le fichier est toujours fermé même si une exception
        }                                // a été levée.
}
```

Après le traitement d'une exception



Les classes d'erreurs Java



- Java définit des classes d'erreur qu'il n'est pas nécessaire d'intercepter ;
- Elles indiquent des erreurs à l'édition de lien par exemple.