Chapitre 15

Gérer les exceptions

Gérer les exceptions

- Le langage java propose des outils de capture des erreurs afin de les traiter directement à l'intérieur des méthodes susceptibles de les détecter, et éviter ainsi que le programme ne s'interrompe.

Exemple sans gestion d'exception

```
public class TestException {
    public static void main(java.lang.String[] args) {
        int i = 3;
        int j = 0;

        System.out.println("résultat = " + (i / j));
        System.out.println("Suite du programme");
    }
}
```

Affiche:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at TestException.main(TestException.java:6)
```

Et le programme s'interrompt.

Exemple avec gestion d'exception

```
public class TestException {
    public static void main(java.lang.String[] args) {
        int i = 3;
        int j = 0;
        try {
           System.out.println("résultat = " + (i / j));
        } catch (ArithmeticException e) {
           System.out.println(e.getMessage());
        }
        System.out.println("Suite du programme");
    }
}
```

Affiche :
/ by zero
Suite du programme

Le programme se poursuit.

Les instructions qui composent le bloc **try** (« essayer ») sont exécutées

- Si aucune erreur ne se produit dans ce bloc, alors le programme se poursuit.
- Si une erreur se produit, alors les instructions placées dans le bloc **catch** (capture) sont exécutées (l'erreur doit être du même type que celle mise entre parenthèses après le catch).

Classes d'exceptions

Les exceptions qui peuvent être levées sont des objets instances de la classe **Exception** et de ses classes dérivées :

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalClassFormatException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, IOException, JAXBException, JMException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SOAPException, SQLException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URIReferenceException, URISyntaxException, UserException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html

Exemple

```
int [] tableau = new int [10] ;

try {
   for( int i = 0; i<20; i++)
      System.out.println(tableau[i]);
} catch( IndexOutOfBoundsException e){
   // traitement de l'exception de dépassement d'indice de tableau
}
```

Messages d'erreur

```
public class TestException {
    public static void main(java.lang.String[] args) {
        int i = 3;
        int j = 0;
        try {
           System.out.println("résultat = " + (i / j));
        } catch (ArithmeticException e) {
           System.out.println(e.getMessage());
           System.out.println(e.toString());
           e.printStackTrace();
        }
    }
}
```

getmessage :
/ by zero

toString :
java.lang.ArithmeticException: / by zero

printStackTrace :
java.lang.ArithmeticException: / by zero
        at TestException.main(TestException.java:6)

Exceptions multiples dans une clause catch

Problème : Il n'est pas rare d'avoir à dupliquer les mêmes lignes de code dans le bloc de code de plusieurs clauses catch().

```
try {
  // traitements pouvant lever les exceptions
} catch(ExceptionType1  e1) {
  // Traitement de l'exception
} catch(ExceptionType2  e2) {
  // Traitement de l'exception
} catch(ExceptionType3  e3) {
  // Traitement de l'exception
}
```

Une solution utilisée pour éviter cette duplication est de catcher un super-type d'exception, généralement le type Exception, mais ce traitement s'appliquera à toutes les exceptions filles et englobera peut-être des exceptions qui auraient nécessité un traitement particulier.

```
try {
  // traitements pouvant lever les exceptions
} catch(Exception e) {
  // Traitement de l'exception
}
```

Solution : On peut déclarer les exceptions dans une même clause catch en les séparant par le caractère "|" :

```
try {
  // traitements pouvant lever les exceptions
} catch(ExceptionType1|ExceptionType2|ExceptionType3 ex) {
  // Traitement de l'exception
} catch(ExceptionType4|ExceptionType5 ex) {
  // Traitement de l'exception
}
```

Clause « *finally* »

```
try {
    // traitements pouvant lever les exceptions
} catch(ExceptionType1 ex) {
    // Traitement de l'exception
} catch(ExceptionType2 ex) {
    // Traitement de l'exception
}
finally {
    // Instructions
}
```

La clause **finally** est toujours exécutée, qu'il y ait eu une levée d'exception ou non. Elle est aussi exécutée si un catch ou le bloc try se termine par un return, un break ou un continue.

throws

Une méthode qui ne traite pas une exception levée par des appels à d'autres méthodes doit donner la liste des exceptions qu'elle peut déclencher avec le mot clé **throws** :

```
void methode() throws E1, E2 {
   // du code qui peut lever
   // des exceptions de la classe E1 ou E2
   // sans les capturer
}
```

Exemple de manipulation de fichier :

```
import java.io.*;

public class testFichier {
    public static void main(String[] args) {
        BufferedReader fR;

        File f = new File("un_fichier.txt");
        fR = new BufferedReader (new FileReader(f));
        String chaine = fR.readLine();
        fR.close() ;
    }
}
```

Erreurs de compilation :

```
testFichier.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be
thrown
                fR = new BufferedReader (new FileReader(f));
                                         ^
testFichier.java:9: error: unreported exception IOException; must be caught or declared to be thrown
                String chaine = fR.readLine();
                                   ^
testFichier.java:11: error: unreported exception IOException; must be caught or declared to be thrown
                fR.close();
                   ^
```

Explication :

Les méthodes FileReader(), readLine(), close() annoncent qu'elles peuvent déclencher des exceptions avec **throws**, il faut donc les traiter avec **try/catch**.

Voir les déclaration de :

public FileReader(File file) **throws FileNotFoundException**
public String readLine() **throws IOException**
public void close() **throws IOException**

Dans :

https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html
https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html

Erreurs de compilation :

```
testFichier.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
                        fR = new BufferedReader (new FileReader(f));
                                                 ^
testFichier.java:9: error: unreported exception IOException; must be caught or declared to be thrown
                        String chaine = fR.readLine();
                                        ^
testFichier.java:11: error: unreported exception IOException; must be caught or declared to be thrown
                        fR.close();
                           ^
```

Solution :

```
import java.io.*;

public class testFichier {
    public static void main(String[] args) {
        BufferedReader fR;

        try
        {
            File f = new File("un_fichier.txt");
            fR = new BufferedReader (new FileReader(f));
            String chaine = fR.readLine();
            fR.close() ;
        }
        catch (IOException e)
        {
            System.out.println("Erreur : " + e.getMessage() );
        }
    }
}
```

Autre solution :

```
import java.io.*;

public class testFichier {
    public static void main(String[] args) throws IOException  {
        BufferedReader fR;

            File f = new File("un_fichier.txt");
            fR = new BufferedReader (new FileReader(f));
            String chaine = fR.readLine();
            fR.close() ;
    }
}
```

throw

Permet de lancer une exception.

```
public class MaClasse {
  static void testAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Accès interdit aux mineurs.");
    }
    else {
      System.out.println("Accès autorisé");
    }
  }

  public static void main(String[] args) {
    testAge(15);
  }
}
```